UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior - Leganés

INGENIERÍA DE TELECOMUNICACIÓN

Proyecto Fin de Carrera

IMPLEMENTACIÓN DE UN SISTEMA DE COMUNICACIONES OFDM IEEE 802.11a SOBRE UN DISPOSITIVO SFF SDR

AUTORA: Lara Rojo Celemín.
TUTORA: Ana García Armada.

Leganés, 2014

Título:	*Implementación de un sistema de comunicaciones OFDM IEEE 802.11a en un dispositivo SFF SDR.*
Autora:	Lara Rojo Celemín.
Tutora:	Ana García Armada.

La defensa del presente Proyecto Fin de Carrera se realizó el día 24 de Octubre de 2014, siendo calificada por el siguiente tribunal:

Presidenta:	Matilde Sánchez Fernández
Secretario:	Víctor Monzón Baeza
Vocal:	Marta Portela García

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente	**Secretario**	**Vocal**

Resumen

El crecimiento de las comunicaciones móviles ha sido vertiginoso durante los últimos años, dando lugar a técnicas cada día más robustas y con mejores prestaciones. Un claro ejemplo es la Multiplexación Ortogonal en Frecuencia (OFDM). Desde sus inicios, este esquema de modulación ha sido objeto de numerosas investigaciones, dando lugar a una de las técnicas más empleadas en comunicaciones móviles.

En este Proyecto Final de Carrera, se pretende implementar un sistema de comunicaciones OFDM basado en el estándar IEEE 802.11a sobre un dispositivo radio. Se trata de prototipar una señal OFDM, transmitirla por un canal radio y recuperar la información enviada.

La novedad de este trabajo reside en la sincronización entre el transmisor y el receptor, además de un bloque ecualizador capaz de corregir todos los errores provocados por el canal radio y que degradarán significativamente la señal transmitida.

Asimismo, además de una serie de simulaciones de cada uno de los bloques del sistema implementado, se mostrarán las tramas de datos OFDM transmitidas y recibidas, comprobando de esta manera el funcionamiento del sistema de comunicaciones.

Palabras clave: Sistema de comunicaciones, OFDM, WLAN Transmisor, Receptor, Sincronización, Estimación de canal, Ecualización, FPGA, VHDL.

Abstract

During the last years, mobile communications have shown a strong growth, giving way to more robust new techniques with better performance. A clear example of this is OFDM (Orthogonal Frequency Division Multiplexing). Since its very beginnings, this modulation scheme has been the object of several studies which have led to one of the most used techniques in mobile communications.

This paper is focused on the implementation of an OFDM communications system based on the IEEE 802.11a standard in a radio device. It is a prototype of an OFDM signal which will be transmitted by a radio channel and finally recovered by the receiver.

What it is new in this paper is the synchronization between the sender and the receiver. Morever a channel equalization has been implemented, correcting all the mistakes due to the radio channel which could degrade significantly the transmitted information.

Furthermore, some simulations of every part of the communications system will be shown, as well as the transmitted and received data framing, allowing the testing of the system performance.

Key words: Communitacions Sytem, OFDM, WLAN, Transmitter, Receiver, Synchronization, Channel Estimation, Equalization, FPGA, VHDL.

Índice general

Índice de figuras

Introducción

En los últimos años, las comunicaciones radio han ido evolucionando a un ritmo muy rápido al igual que su uso y demanda, los cuales también han experimentado un enorme crecimiento.

En el mundo actual, queremos estar continuamente comunicados hasta tal punto que no podemos vivir sin estar conectados. Esta necesidad no sólo ha sido clave en la evolución que han sufrido las tecnologías de comunicación inalámbrica sino también en las técnicas y en los dispositivos que las implementan.

La Multiplexación Ortogonal en Frecuencia (OFDM) ha sido una de esas técnicas que han sufrido ese auge. Desde sus inicios, esta modulación ha ido evolucionando en cuanto a robustez, velocidad y eficiencia, dando lugar a uno de los esquemas de modulación más utilizado en las comunicaciones móviles actuales. Sus principales aplicaciones se encuentran en sistemas de comunicaciones como WiFi, WiMAX, ADSL, DVB-T y telefonía 4G entre otros.

Debido precisamente al avance de estas tecnologías, es necesario el desarrollo de procesadores capaces de soportar todo este procesamiento de señal. Las plataformas de desarrollo basadas en FPGAs (Field Programmable Gate Array) han sido una de las que más ha evolucionado, experimentado mejoras en velocidad, capacidad de procesamiento y almacenamiento.

El dispositivo SFF SDR (Small Form Factor Software Defined Radio) es una placa que combina además de un módulo de procesamiento de señal formado por una FPGA y un DSP (Digital Signal Processing), un módulo de conversión de datos y un módulo de radiofrecuencia. Gracias a estos dos últimos módulos, seremos capaces de transmitir y recibir señales dentro de un gran rango de frecuencias, lo que hace que este dispositivo pueda ser utilizado para una infinidad de aplicaciones que van desde lo militar hasta lo comercial.

Motivación

La motivación de este proyecto es poder llevar a cabo una implementación hardware utilizando precisamente este tipo de dispositivos que combinan tanto la lógica programable como el procesamiento digital de las señales, utilizando lenguajes de alto nivel como VHDL y C. Además no sólo seremos capaces de implementar una señal OFDM sino que también podremos transmitirla de forma radio gracias a los módulos de radiofrecuencia y de conversión de datos de nuestro dispositivo, pudiendo así comprobar los efectos de un canal de comunicaciones radio sobre la información transmitida.

Objetivos

Este proyecto, al igual que se hizo en *Implementación de un sistema de comunicación en un dispositivo radio* [3], busca darle continuidad al trabajo realizado en *Implementación de un sistema OFDM en un dispositivo SFF SDR* [4].

El principal objetivo de este trabajo es la realización de un receptor que sea capaz de recuperar la señal original, sincronizando la trama y corrigiéndola de los errores introducidos por el canal mediante la ecualización.

También se pretende mejorar la implementación previamente realizada, mejorando el número de recursos y la latencia del sistema.

Trabajos Previos

Este trabajo busca la implementación en VHDL del trabajo realizado en *Design and Implementation of Synchronization and AGC for OFDM-based WLAN Receivers* [5], además de dar continuidad a los Proyectos Fin de Carrera *Implementación de un sistema OFDM en un dispositivo SFF SDR* [4] e *Implementación de un sistema de comunicación en un dispositivo radio* [3].

En el primero de ellos, se realiza una implementación de un transmisor OFDM. En el segundo, en cambio, se explica el diseño e implementación de un demodulador en cuadratura.

Este trabajo busca continuar con el trabajo realizado previamente, mejorándolo y uniendo ambos trabajos, además de desarrollar un receptor OFDM con el fin de recuperar la información enviada por el transmisor.

Contexto

Este proyecto fin de carrera se ha realizado en el departamento de Teoría de la Señal y Comunicaciones de la Universidad Carlos III de Madrid y se encuentra enmarcado dentro de una línea de trabajo que busca poner en práctica algunos de los trabajos realizados por los profesores del departamento con el fin de poder desarrollar una implementación práctica de los mismos, consiguiendo así probar sus investigaciones.

Particularmente este proyecto se sitúa entre aquellos proyectos que buscan la implementación práctica de sistemas de comunicación multiportadora usando un dispositivo SFF SDR.

Las pruebas del mismo se han realizado en el laboratorio de sistemas de comunicaciones para seguridad y espacio del Centro Mixto EADS del Parque Científico de la Universidad Carlos III de Madrid.

Contenido de la memoria

A continuación, se detalla brevemente la estructura que va a seguir la memoria de este Proyecto Fin de Carrera:

Capítulo 1: OFDM. En este capítulo se explican brevemente los conceptos básicos de esta modulación, mostrando además un sistema de comunicaciones OFDM comentando cada uno de los módulos que lo componen.

Capítulo 2: Transmisor. En él se presenta un transmisor OFDM. A lo largo de este capítulo se expondrá todo el proceso de implementación del mismo, comentando con todo detalle cada uno de los bloques y sus diferencias con respecto al caso teórico. Por último, se mostrarán una serie de simulaciones en las que se podrá comprobar que dicho esquema es capaz de generar la trama de datos a enviar.

Capítulo 3: Receptor. En este capítulo se expone el receptor OFDM. Como ya se hizo en el capítulo 2, se detallará el proceso de implementación que concluirá con una simulación del mismo.

Capítulo 4: Sincronismo. Aunque el sincronismo forma parte del receptor, debido a la complejidad de esta operación se muestra por separado. En este capítulo se muestra el algoritmo empleado en el proceso de sincronización temporal de la señal, mostrando una serie de simulaciones del mismo.

Capítulo 5: Ecualización. Al igual que el capítulo anterior, el proceso de ecualización debido a su alto grado de dificultad se ha preferido mostrarlo de forma independiente. En él, se explica la técnica escogida tanto para la estimación de canal como para la interpolación, verificando los resultados obtenidos mediante simulación.

Capítulo 6: Pruebas y Resultados. En este capítulo se muestran los resultados obtenidos de la implementación de nuestro sistema de comunicaciones, analizando los mismos.

Capítulo 7: Conclusiones. Se establecen las conclusiones además de las líneas futuras y las dificultades encontradas a lo largo de la realización de este Proyecto Final de Carrera.

Capítulo 1

OFDM

En este capítulo se va a describir brevemente la modulación OFDM empleada en este PFC. En primer lugar, se expondrán los conceptos fundamentales, mostrando así una visión rápida de la misma. También se hablará de sus principales campos de aplicaciones con el fin de mostrar los motivos que han impulsado a implementar esta modulación. Después, se comentará el sistema OFDM a implementar, explicando además de las técnicas empleadas para la eliminación de las interferencias, el sincronismo y la estimación de canal.

1.1. Principios básicos

La OFDM consiste en la multiplexación de un conjunto de portadoras ortogonales entre sí. Este tipo de modulación multiportadora es frecuentemente utilizada en esquemas de comunicaciones de banda ancha como ADSL, DVB-T, WiMAX, telefonía 4G, entre otros. Su objetivo es el de transmitir y recibir datos a la mayor velocidad posible, intentando además conseguir la mínima tasa de error (BER) posible.

La transmisión está basada en el uso de múltiples portadoras lo que se consigue dividiendo el flujo de datos con tasa de símbolo R_s en N subflujos de tasa R_s/N, de tal forma que cada uno de estos subflujos modulará una subportadora con un ancho de banda B_s/N. De ahí que se pueda decir que se trata de la transmisión de N símbolos en N subportadoras en paralelo. En cuanto al tiempo de símbolo T_s, se verá aumentado en un factor N, dando lugar a un periodo de símbolo $N \cdot T_s$.

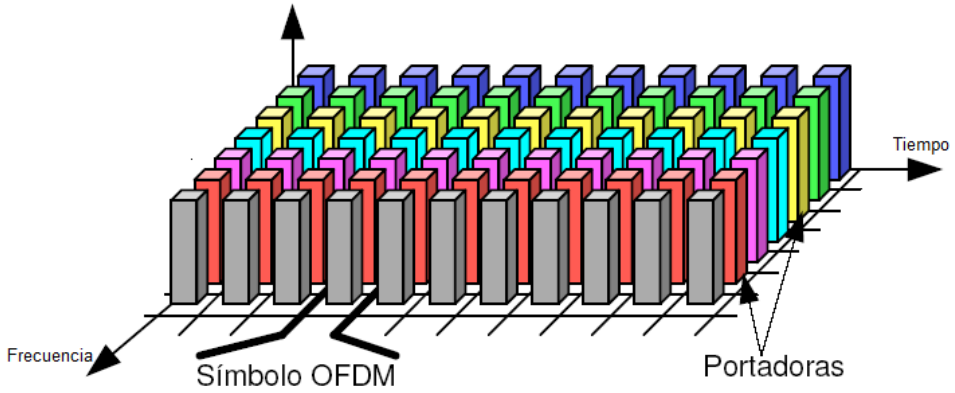


Figura 1.1: Trama OFDM en tiempo y en frecuencia

Al dividir el canal en N subcanales, se consigue pasar de un canal de banda ancha y selectivo en frecuencia, a varios subcanales de banda estrecha (figura 1.1) y con respuesta plana en frecuencia. Sin embargo, a pesar de conseguir dichas características también se está aumentando el tiempo de símbolo, por lo que también aumenta la probabilidad de que se produzca *fast fading*, de ahí que se tenga que llegar a un compromiso.

La principal ventaja de la OFDM frente a otros esquemas de modulación como la FDM es la ortogonalidad de las portadoras. La ortogonalidad se consigue gracias a la separación entre portadoras adyacentes. Dicha separación es tal que, mientras una subportadora está activa, el resto permanecerán inactivas al tener un nulo en el centro de la subportadora adyacente como se muestra en la figura 1.2.

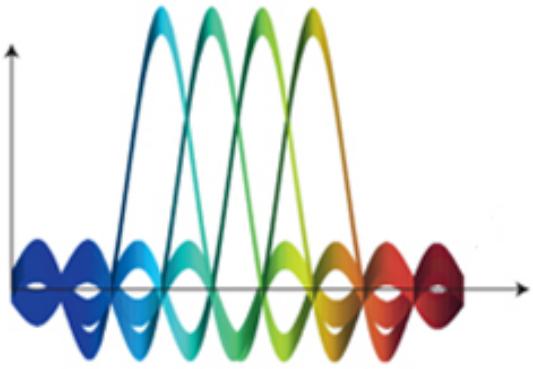


Figura 1.2: Espectro señal OFDM

Otra de las ventajas de la OFDM como se verá a continuación es su fácil y eficiente implementación gracias al uso de los algoritmos FFT/IFFT (Transformada Rápida de Fourier). Además su elevada eficiencia espectral al usar múltiples portadoras hace que este esquema sea bastante utilizado.

Por otra parte, la OFDM cuenta también con una serie de desventajas como puede ser la sincronización ya que, como se explicará en la sección 1.5, jugará un papel fundamental en el proceso de recuperación de la señal transmitida.

Por otra parte, hay que tener en cuenta que para evitar los efectos de las interferencias se insertan unos intervalos de guarda y prefijos cíclicos consiguiendo de esta forma mantener la orgonalidad entre las portadoras, lo que supone una pérdida de eficiencia.

1.2. Sistema OFDM

Dado que el desarrollo matemático de la OFDM está descrito en anteriores fases del proyecto [3] [4], se pasará directamente a estudiar la transmisión y recepción.

La imagen de la figura 1.3 muestra el sistema OFDM completo. Dicho esquema está basado en [5]. A lo largo de esta memoria, se explicará con más detalle cada uno de los bloques que forman parte tanto del transmisor como del receptor implementado. También se comentará toda la problemática debida a las limitaciones del hardware en el diseño de ambas partes.

En primer lugar, hay que suponer un generador de bits, el cual produce el flujo binario *a*. Esos bits pasan por un conversor serie-paralelo para ser modulados posteriormente usando una determinada constelación. Tras el modulador, se tendrá un conjunto de símbolos como los de la ecuación 1.1.

$$A[m] = \{\ A_0[n]\ A_1[n]\ ...\ A_{N-1}[n]\ \} \tag{1.1}$$

Una vez generados los símbolos, se realiza la IFFT obteniendo la ecuación 1.2.

$$S(t) = \sum_{k=0}^{N-1} A_k e^{j2\pi kt/T} \tag{1.2}$$

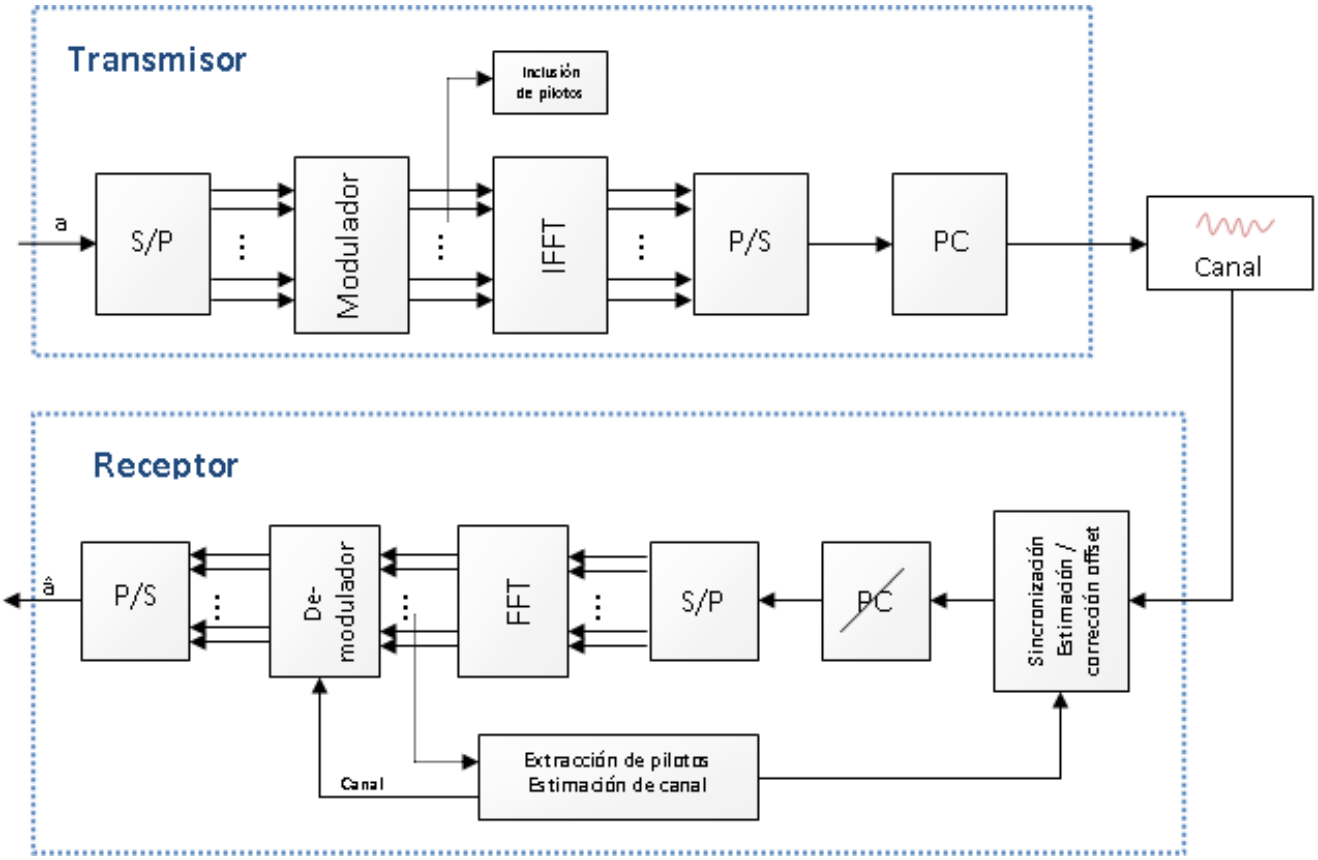


Figura 1.3: Esquema del sistema OFDM

Antes de pasar al dominio temporal los símbolos, se insertan los pilotos. Se trata de unas señales de referencia que, como se verá en las siguientes secciones, serán utilizadas en el proceso de estimación de canal. Tanto su ubicación como su uso en el proceso de estimación de canal se comentará más adelante, en la sección 1.4.

Una vez realizada la IFFT, la información será convertida de nuevo a formato serie.

Por último, se incluye el Prefijo Cíclico (PC). Al igual que los pilotos se trata de unas señales de referencia que se insertan al inicio de cada símbolo y que, como se comentará en la sección 1.3, eligiéndolo de un determinada longitud se pueden evitar los efectos de la Interferencia Intersimbólica (ISI) y, si además se escoge de tal forma que haga nuestro símbolo periódico, se pueden evitar los efectos de la Interferencia entre Portadoras (ICI) al conseguir independizar los canales gracias a la convolución circular.

Una vez generada la trama OFDM, esta es convertida a analógico por el Con-

versor Digital-Analógico (DAC) y modulada por el módulo de radiofrecuencia a la frecuencia de transmisión. Como en todo sistema de comunicaciones, el canal introducirá ruido y otros elementos que degradarán la señal obtenida en el receptor.

Por otra parte, el receptor, tras realizar todas la conversiones en frecuencia hasta obtener la señal de nuevo en banda base, hará la conversión a digital de la señal por medio del Conversor Analógico-Digital (ADC).

Anteriormente se ha mencionado que uno de los principales inconvenientes de la OFDM es la sincronización. Pues como se verá en el capítulo 4, será precisamente este bloque el que resulte crítico a la hora de recuperar la información transmitida. A partir de los preámbulos enviados en la cabecera del estándar implementado, se realiza esta acción. Como hemos dicho, más adelante, en el capítulo 4 se detallará el procedimiento ejecutado.

Una vez sincronizada la trama, el receptor ya es capaz de iniciar las operaciones necesarias para la obtención de los bits, por tanto, comenzará con la extracción del prefijo cíclico seguida de la FFT.

Tras la conversión de los símbolos de nuevo al dominio de la frecuencia, se utilizarán los pilotos previamente insertados para corregir los posibles errores introducidos por el canal mediante la ecualización.

Por último, la señal será demodulada y los símbolos, con ayuda de un decisor, serán transformados en bits, finalizando así el proceso de recepción.

Aunque en el esquema no aparezca, antes de la transmisión de los símbolos hay un modulador IQ con el fin de unir la señal real con la imaginaria, por lo que en nuestro receptor habrá que incluir también un demodulador IQ, cuyo diseño se ha realizado en fases anteriores del proyecto [3].

1.3. Intervalo de guarda y Prefijo Cíclico

Debido al multitrayecto, la señal recibida por el receptor estará compuesta por la superposición de múltiples réplicas, provocando no sólo el solape entre símbolos adyacentes sino también entre las muestras del mismo símbolo. Estos efectos son conocidos como Interferencia Intersimbólica o ISI e Interferencia entre portadoras o ICI respectivamente.

Con el fin de eliminar estas interferencias se insertan intervalos de guarda como los mostrados en la figura 1.4. Este intervalo se elige de una duración igual al máximo retardo de todas las réplicas del símbolo transmitido. De esta forma, se asegura que todas las componentes del símbolo lleguen antes que el símbolo siguiente, es decir, siempre y cuando el intervalo de guarda sea mayor que el ensanchamiento del retardo del canal no se producirá ISI.

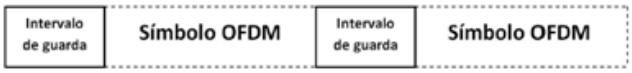


Figura 1.4: Implementación del intervalo de guarda

Sin embargo, con este método no se elimina la ICI, lo que da lugar a una pérdida de la ortogonalidad, degradando significativamente la transmisión. La solución ante esta interferencia es la inclusión de un Prefijo Cíclico (PC). Este prefijo consiste en insertar, tras realizar la IDFT, la parte final del símbolo en la parte inicial del mismo como se muestra en la figura 1.5.

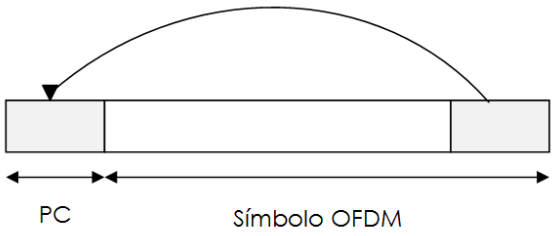


Figura 1.5: Esquema de implementación del PC

La finalidad de la extensión cíclica es conseguir que la respuesta al canal sea una convolución circular en lugar de una lineal, ya que el producto de dos DFT equivale en el dominio del tiempo a una convolución circular.

Con esta técnica, no sólo se consigue que la ICI sea nula sino también la ISI. Esta afirmación será cierta siempre y cuando el tamaño de la extensión cíclica sea mayor que el ensanchamiento del retardo del canal.

La inclusión del PC supone una pérdida de eficiencia energética y del ancho de banda ya que esos bits no se corresponden con información útil. A pesar de ello, dado que se consigue eliminar tanto la ISI como la ICI facilitando así el proceso de recuperación de la información transmitida, se acepta esa pequeña pérdida de eficiencia.

1.4. Pilotos

Para poder recuperar correctamente la señal transmitida, es necesario realizar un proceso de estimación de canal. Para ello, se inserta un patrón como referencia. Dicho patrón está formado por una serie de portadoras de referencia o pilotos.

La distribución de los pilotos a lo largo de la trama debe ser conocida tanto por el transmisor como por el receptor. En la figura 1.6 se puede ver un ejemplo de esta distribución. Como se ve, existen dos separaciones: una en el dominio del tiempo (N_t) y otra en el dominio de la frecuencia (N_f).

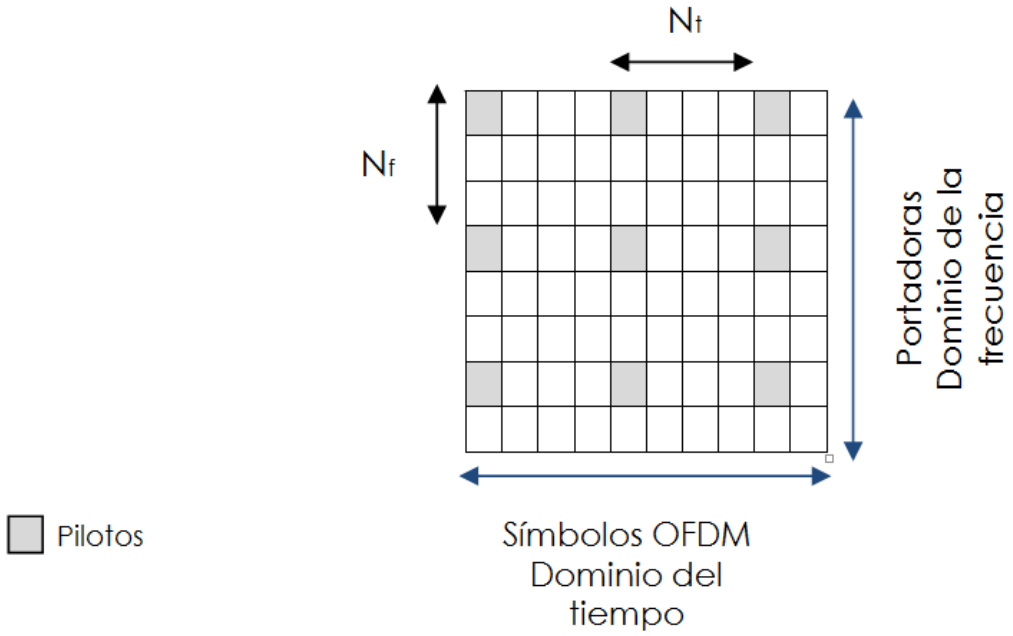


Figura 1.6: Esquema de distribución de pilotos

Como se verá en la sección 1.6, gracias a estas señales de referencia seremos capaces de ecualizar nuestra señal recibida corrigiendo así pequeñas desviaciones en amplitud y en fase debidas al paso de la señal por el canal.

La inclusión de estos pilotos supone una pérdida en la eficiencia espectral pues se están transmitiendo señales que no contienen información útil. Es cierto, que cuantos más pilotos se incluyan en la trama, mejor será la estimación de canal. Sin embargo, como se ha dicho la eficiencia se verá afectada significativamente. Por ello, hay que encontrar una situación de compromiso entre la eficiencia y la calidad de la estimación de canal.

1.5. Sincronización

En todo sistema de comunicación la sincronización juega un papel fundamental, pues gracias a ella seremos capaces de recuperar correctamente la información.

Una falta de sincronización, tanto en tiempo como en frecuencia o en fase, daría lugar a una serie de errores en la demodulación de la señal recibida. Para empezar, una pequeña desviación de nuestra frecuencia provoca la pérdida de la ortogonalidad de la señal OFDM, lo que lleva a un muestreo en frecuencia incorrecto dando lugar a la Interferencia entre portadoras (ICI).

Por otra parte, las desviaciones en fase provocan una rotación de los símbolos de la constelación, haciendo que el decisor realice la asociación símbolo-bits de manera errónea.

Siguiendo las indicaciones del estándar IEEE 802.11a, cada paquete contiene unos preámbulos PLCP (Protocolo de convergencia de capa física), mostrados en la figura 1.7, cuya función, entre otras muchas, es la de sincronización.

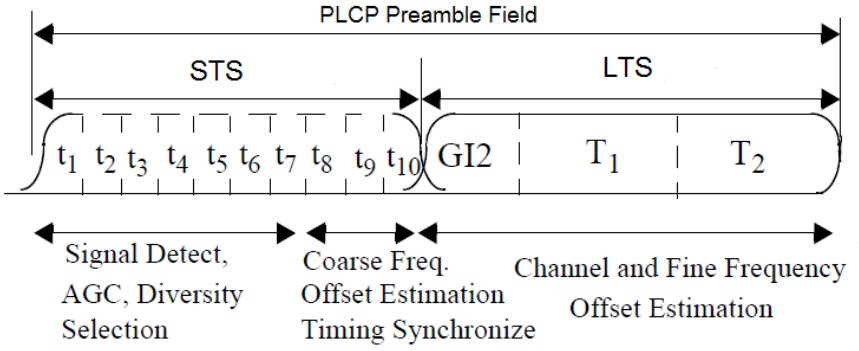


Figura 1.7: Estructura de entrenamiento WLAN [1]

Los preámbulos se componen de dos secuencias de entrenamiento: la secuencia corta de entrenamiento (Short Training Sequence, STS) y la secuencia larga de entrenamiento (Long Training Sequence, LTS), ambas con la misma longitud.

Como se puede ver en la figura 1.7, la STS es usada para la detección de la señal, estabilización del control automático de ganancia (AGC), diversidad (en el caso de múltiples antenas), sincronización en frecuencia, etc.

La STS está formada por la repetición de 10 símbolos cortos de entrenamiento, t_i, con una longitud de 16 bits. En cuanto a la secuencia larga de entrenamiento,

está formada por dos símbolos largos, T_i y sus funciones son la de estimación de canal y de frecuencia.

Ambas secuencias se generan a partir de la modulación de unas secuencias de símbolos especificadas en el estándar [1].

El algoritmo implementado para la sincronización temporal estará basado en el uso de una métrica tal y como se explica en [5]. Más adelante en el capítulo 4 se comentará con más detalle el algoritmo y la métrica anteriormente mencionada.

1.6. Estimación de canal y Ecualización

Como ya sabemos el canal introduce una serie de cambios en los símbolos transmitidos, provocando pequeñas variaciones en amplitud y en fase en nuestra señal, lo que hace necesario un módulo que sea capaz de corregir o al menos minimizar estos posibles errores de la trama de símbolos.

La función del ecualizador es precisamente esa, la de invertir las variaciones introducidas por el canal consiguiendo así una recuperación de la información original sin errores. Para poder realizar esta recuperación se necesita conocer el comportamiento del canal. Por tanto, el proceso de ecualización está basado en una estimación de canal.

Gracias a la OFDM, como hemos ido viendo a lo largo de este capítulo, la estimación de canal se simplifica al tratar cada subportadora de forma independiente, pudiendo calcular la estimación para cada subcanal.

Existen múltiples técnicas de estimación de canal, algunas están basados en la inserción de unas señales de referencia. Otras, en cambio, utilizan métodos estadísticos. Incluso hay un tercer grupo que utiliza ambos métodos.

El primero de ellos es el más simple. Consiste en insertar una serie de pilotos en unas determinadas posiciones de la trama. Tanto el símbolo como la posición son conocidos por el transmisor y el receptor. A partir de estos datos, el ecualizador es capaz de realizar la estimación. Su mayor inconveniente es el de la eficiencia espectral que se verá disminuida a medida que se incluyan más señales de referencia.

Por otra parte, la estimación mediante métodos estadísticos resulta más compleja debido a la dificultad de encontrar un modelo estadístico que se ajuste a las

variaciones introducidas por el canal. A diferencia de la anterior técnica, este método tendrá una mayor eficiencia espectral.

Por último, están las técnicas que combinan ambos métodos, las cuales buscan el compromiso entre eficiencia espectral y complejidad.

Este trabajo se centra en las técnicas basadas en la inserción de pilotos. Por tanto, nuestro propósito es el de buscar esa constante multiplicativa que modifica nuestra señal transmitida.

El ruido afectará significativamente en el cálculo de la estimación, por ello se intenta buscar que los pilotos se correspondan con los símbolos de mayor energía. Dado que la constelación empleada en este trabajo es una 4-QAM, cualquiera de los cuatro símbolos posibles será válido.

En la actualidad, existen diferentes técnicas de estimación de canal basadas en la inserción de pilotos. Las más comunes son LS (Least Squares o mínimos cuadrados) y MMSE (Minimum Mean Square Error o de mínimo error cuadrático medio).

El estimador LS trata de minimizar el error cuadrático entre los símbolos transmitidos y recibidos mediante la ecuación 1.3

$$\hat{H}_{LS} = arg_{H(n)} \min\{(Y(n) - X(n)\dot{H(n)})^2\} \tag{1.3}$$

donde $Y(n)$ es la señal recibida, $X(n)$ la señal transmitida y $H(n)$ nuestro estimador. Desarrollando la ecuación anterior y minimizándola, se llega a la expresión de la ecuación 1.4.

$$\hat{H}_{LS} = \frac{Y(n)}{X(n)} \tag{1.4}$$

Como se ha visto este estimador considera que las señales son deterministas, además no tiene en cuenta la presencia de ruido, de ahí que se prefieran utilizar otras técnicas que sí valoran el ruido y otros factores que distorsionan nuestra señal.

Por su parte, el estimador MMSE resulta bastante más complejo. Se trata de un estimador bayesiano y como tal, supone el conocimiento a priori del comportamiento estadístico del canal. Esta técnica proporciona mejores resultados al ser

más robusto frente al ruido y a la ISI pero debido a su complejidad se ha optado por desarrollar el primero.

Por último, una vez decidido el tipo de estimador, sólo quedan por considerar el tipo de interpolación que usará nuestro estimador. Al igual que ocurría con las técnicas de estimación, se puede elegir entre distintos tipos de interpoladores. Este trabajo se ha centrado en dos: en la interpolación lineal y en la interpolación usando la IDFT/DFT.

La interpolación lineal consiste en calcular la recta que une dos puntos conocidos, aproximando así los puntos comprendidos entre esos dos valores conocidos, ecuaciones 1.5 y 1.6. La dificultad de este tipo de interpolación reside en el cálculo de n, lo que se soluciona a partir del *Teorema de Tales* dando lugar a la expresión de la ecuación 1.7.

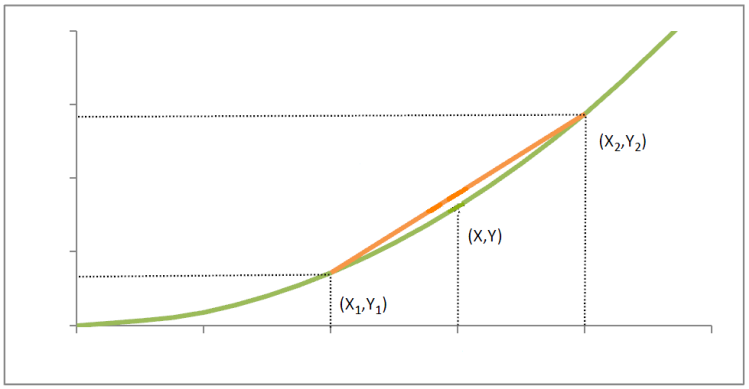


Figura 1.8: Representación interpolación lineal

$$y = m \cdot x + n \tag{1.5}$$

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + n \tag{1.6}$$

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1) + y_1 \tag{1.7}$$

La interpolación lineal es una técnica bastante simple. Sin embargo, no ofrece una gran precisión, lo que se consigue con interpoladores de segundo orden o filtros de Respuesta Finita al Impulso (FIR).

En cambio, la interpolación utilizando la IDFT/DFT consiste en el uso de la Transformada de Fourier Discreta junto con la inserción de ceros. En los últimos años, este método se ha ido utilizando debido entre otras cosas a la mejora de los algoritmos FFT/IFFT.

En primer lugar, se pasa al dominio del tiempo todas las señales piloto. Tras la IFFT, se procede a la inserción de ceros, tantos como portadoras tenga el sistema OFDM y, por último, se vuelve a transformar al dominio de la frecuencia, consiguiendo así la interpolación. Este tipo de interpolación se puede considerar como un filtrado paso bajo ya que se están anulando las componentes de alta frecuencia mediante la inserción de ceros.

Con todo lo explicado anteriormente, ya se puede ecualizar nuestra trama de datos, corrigiendo así esas desviaciones introducidas por el canal.

Por otra parte, no se ha tenido en cuenta la información enviada en los preámbulos. Como se vio en la sección 1.5, la cabecera del estándar 802.11a consta de dos secuencias de entrenamiento, una corta y otra larga. Es precisamente la LTS la que se ocupa de la estimación de canal entre otras funciones. Su estudio e implementación no se realiza en este trabajo, dejándolo para futuras fases del mismo.

Obviamente, si se utilizan los resultados de la estimación a partir de los pilotos junto con la estimación basada en los símbolos largos de entrenamiento, se conseguirá realizar una estimación de canal mucho más precisa, aprovechando además todos los recursos del sistema.

Capítulo 2

Transmisor

En este capítulo se presenta el diseño e implementación del transmisor OFDM, explicando claramente todas las modificaciones sufridas con respecto a otras fases del proyecto llevadas a cabo en [3] y [4].

Como ya se ha hecho en anteriores fases, se detallarán cada uno de los bloques que forman parte del transmisor implementado con el fin de entender este bloque del sistema en su totalidad.

Por último, una vez implementado en VHDL el diseño, se mostrarán las simulaciones de dichos bloques pudiendo comprobar así su correcto funcionamiento.

2.1. Descripción del sistema

En la sección 1.2 se explicó el sistema OFDM que se pretende implementar. A continuación se muestra de nuevo en la figura 2.1 el esquema del bloque transmisor.

La implementación de nuestro sistema variará significativamente con respecto al esquema propuesto en la sección 1.2, realizando una versión simplificada del mismo.

En primer lugar, nuestro diseño no cuenta con un generador de bits aleatorio, sino que por simplicidad se transmite de manera continua un vector de bits de tamaño arbitrario.

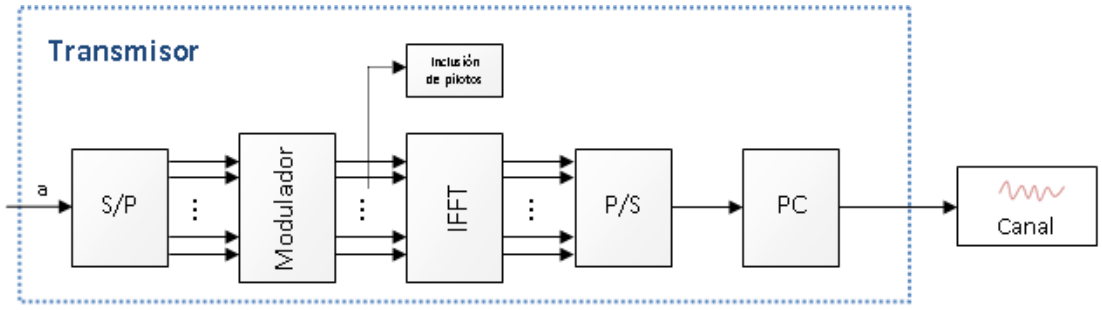


Figura 2.1: Estructura del bloque transmisor

A continuación, se encuentra el bloque serie-paralelo, el cual no será implementado en nuestro modelo pues, como iremos comprobando en la descripción de los bloques de nuestro sistema, las operaciones FFT/IFFT se hacen en serie, prescindiendo así de este módulo.

Debido a la ausencia del conversor serie-paralelo, la modulación se realiza también en serie, provocando no sólo la latencia de nuestro sistema sino también la necesidad de incorporar memorias RAM para almacenar los símbolos generados.

La modulación escogida es una 4-QAM dando lugar a cuatro posibles símbolos, los cuales constan de una parte real y otra imaginaria. Dado que VHDL no trabaja con señales imaginarias, hace que sea necesario el uso de dos memorias RAM para guardar estos símbolos: una de ellas para la parte real y la otra, para la parte imaginaria.

Antes de realizar la IFFT, hay que insertar los pilotos en unas determinadas posiciones y así tener información adicional para poder estimar el canal para poder corregir errores provocados por el canal gracias a la ecualización. Esta operación se realizará a la vez que se guardan los datos en las memorias RAM, consiguiendo así un mejor uso de los recursos del dispositivo.

Una vez conseguidos los primeros N símbolos del sistema, se calcula la IFFT de N muestras. El símbolo OFDM conseguido es guardado en otra memoria RAM con lo que, al igual que ocurría con la anterior RAM, es necesario la implementación de dos memorias para los símbolos OFDM reales e imaginarios.

Cuando estén calculados todos los símbolos OFDM, ya se puede iniciar la transmisión de la trama. Para ello, hay que enviar al DAC del dispositivo los valores de las componentes reales e imaginarias y el propio DSP es capaz de realizar la modulación en cuadratura, originando una única señal que es transmitida a través de la antena a 260 MHz.

El esquema de la figura 2.1 se corresponde con un diagrama de bloques de un transmisor de un sistema OFDM. Sin embargo, el sistema que se está implementando, como se ha ido viendo, está basado en el estándar WLAN IEEE 802.11a, en el cual cada trama consta de una cabecera con información adicional para la detección, sincronización y estimación de canal. El formato y el tamaño de esta cabecera ya se explicó en la sección 1.5.

Por tanto, como se ha comentado, lo primero que se transmite es la cabecera que, como se explicará en el capítulo 4, en nuestro diseño sólo se implementará la secuencia corta de entrenamiento con el fin de encontrar el sincronismo temporal. La ventaja de esta STS es que es constante, permitiéndo así ahorrar su cálculo en cada trama.

Tras enviar los preámbulos al DAC, comienza la transmisión de los símbolos OFDM pero antes hay que insertar los prefijos cíclicos de cada uno de ellos. El tamaño del prefijo escogido es de $N/8$, por lo que las $N/8$ últimas muestras del símbolo OFDM serán repetidas al inicio del mismo.

Una vez transmitido el último símbolo OFDM, se iniciará la generación de una nueva trama OFDM.

Con todo lo anteriormente descrito y para los objetivos planteados en este proyecto, el esquema de la figura 2.1 se traduce en el representado en la figura 2.2.

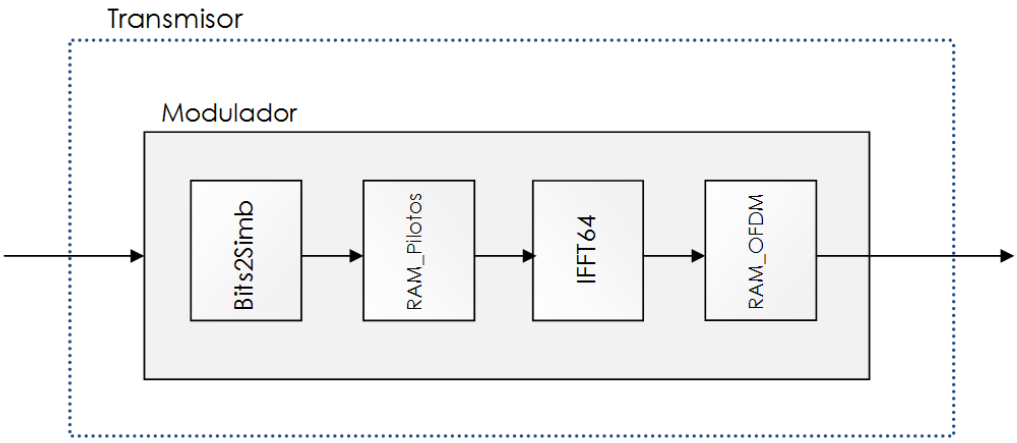


Figura 2.2: Diagrama de bloques del sistema transmisor

Como se puede ver en la figura 2.2, el bloque *Transmisor* consta de un único bloque *modulador*, el cual a su vez está formado por otros cuatro bloques: *bit2Simb*, *RAM_Pilotos*, *IFFT64* y *RAM_OFDM*. Todos ellos hacen que sean posibles las

operaciones anteriormente descritas. Sin embargo, para entender mejor su funcionamiento, se irá detallando cada uno de estos bloques por separado, comenzando por los de nivel jerárquico inferior.

Antes de explicar cada uno de los bloques del sistema, dado que se irán mostrando las simulaciones de cada uno de los bloques anteriormente mencionados, es importante dejar claro el valor de algunos elementos de nuestro sistema. Estos valores se emplearán tanto en las pruebas como en la simulación:

- `N_OFDM`. El número de símbolos OFDM por trama será de 100
- `N`. El número de portadoras será 64
- `DataInSize`. La longitud de cada símbolo será de 16 bits
- N_t. La separación temporal de los pilotos se establecerá en 4
- N_f. La separación frecuencial de los pilotos es de 8

2.2. Bits2Simbolos

Diseño

Este bloque es el encargado de convertir los bits que van llegando en símbolos, es decir, su cometido es el de la asociación de símbolos en función de una constelación, en este caso, una 4-QAM.

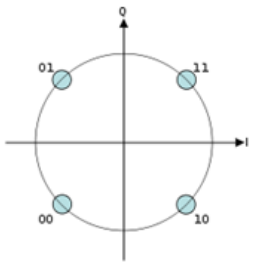


Figura 2.3: Constelación 4-QAM

Aunque este bloque está diseñado para poder codificar los bits a partir de diferentes constelaciones, actualmente sólo está implementada la 4-QAM, en la cual,

como se puede ver en la figura 2.3, sigue una codificación de tipo Gray para la transformación bit-símbolo.

Finalmente, el diseño del bloque *Bits2Simbolos* queda como se muestra en la figura 2.4. Como vemos consta de la siguientes interfaces de entrada y salida.

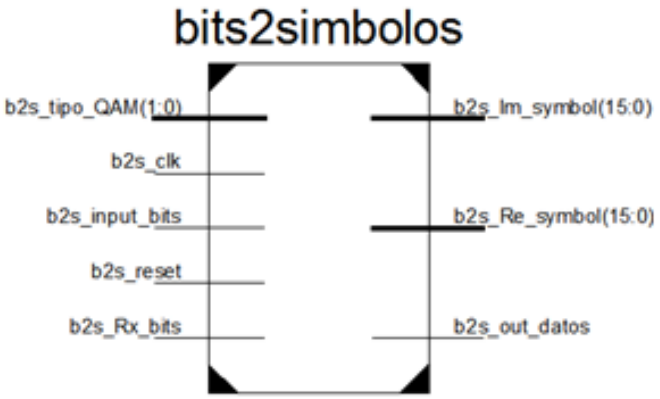


Figura 2.4: Diagrama de bloques de *Bits2Simbolos*

Interfaces de Entrada:

- `b2s_clk`: reloj del sistema
- `b2s_reset`: señal de reset del sistema. Activa a nivel alto
- `b2s_Rx_bits`: señal de habilitación del bloque
- `b2s_tipo_QAM`: 4, 8, 16 ó 64 QAM
- `b2s_input_bits`: bits de entrada

Interfaces de salida:

- `b2s_out_datos`: señal para indicar que el símbolo se ha generado
- `b2s_Re_symbol`: Parte real del símbolo
- `b2s_Im_symbol`: Parte imaginaria del símbolo

Funcionamiento y simulación

Su funcionamiento es bastante sencillo. Al tratarse de una constelación 4-QAM se necesitan dos bits para poder generar el símbolo, por lo que este bloque va a generar un retardo de un ciclo de reloj por cada símbolo generado. Esto se puede comprobar observando la figura 2.5.

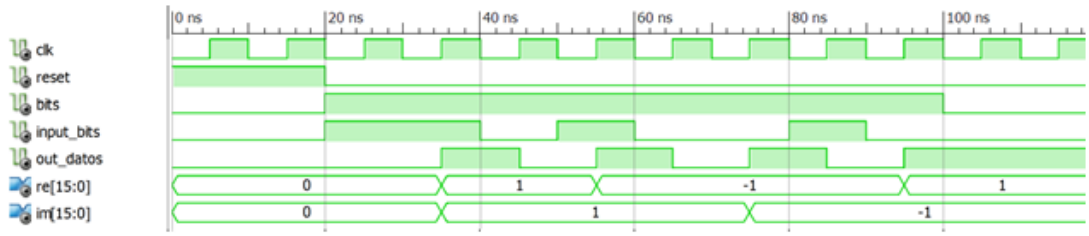


Figura 2.5: Simulación del bloque *Bits2Simbolos*

Además del retardo generado, también se puede comprobar en la simulación de la figura 2.5, cómo la asociación de símbolos se realiza una vez que la señal de habilitación del bloque está activada (`bits`). Dicha asociación se hace con cada uno de los cuatro símbolos posibles de la constelación seleccionada.

2.3. RAM_Pilotos

Diseño

La trama OFDM implementada está compuesta por N_OFDM símbolos OFDM con un total de N portadoras. Por tanto, por cada símbolo OFDM se necesitan generar N símbolos 4-QAM. Por ello, es necesaria la inserción de memorias que vayan almacenando dichos símbolos.

Sin embargo, para reducir tanto el número de recursos como el tiempo de procesamiento de las tramas, estas memorias no sólo cumplen la función de almacenar, sino también la de inserción de los pilotos.

Como ya se explicó en la sección 1.4, existen dos tipos de espaciado de pilotos: en tiempo, N_t, y en frecuencia, N_f. El primer problema encontrado con este espaciado es la imposibilidad de seleccionar un valor arbitrario de ambos valores.

Teniendo en cuenta que la inserción de pilotos se realiza para una futura estimación de canal, la cual implica una interpolación, en este caso lineal, hace que sea inevitable la inserción de pilotos en el primer y en el último símbolo de la trama.

Como se verá, tanto las pruebas como las simulaciones se han realizado con unas tramas formadas por $N\_OFDM = 100$ símbolos OFDM con un total de $N = 64$ portadoras. Por tanto, dado que tanto el símbolo 1 como el 100 deben contener portadoras piloto, los posibles valores para la separación temporal de los pilotos son: 3, 9, 11 *y* 33.

En cuanto al valor del espaciado en frecuencia, este viene limitado por el tamaño de la FFT y, como se verá a continuación en la sección 2.4, no podremos escoger cualquier valor de la FFT, sino que esta debe cumplir que el número de puntos sea potencia de dos.

En la figura 2.6, se puede ver cómo queda el diseño del bloque.

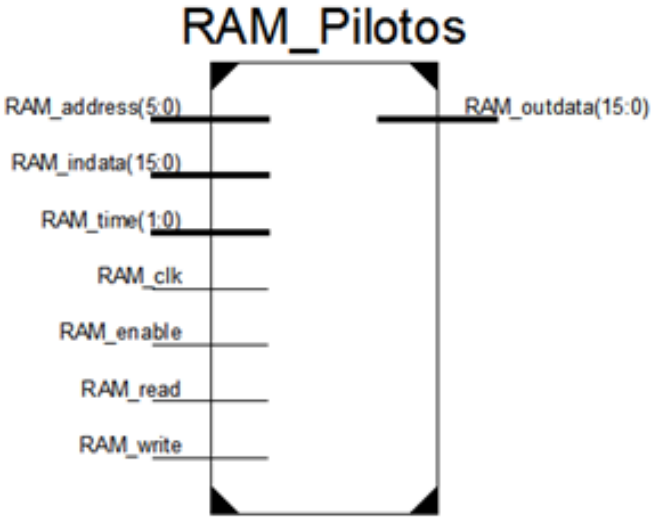


Figura 2.6: Diagrama de bloques de *RAM_Pilotos*

Interfaces de entrada:

- `RAM_clk`: señal de reloj del sistema
- `RAM_enable`: señal de activación
- `RAM_write`: habilitar escritura
- `RAM_read`: habilitar lectura
- `RAM_address`: posición de memoria para lectura/escritura
- `RAM_time`: contador para el espaciado temporal de los pilotos
- `RAM_inData`: dato a guardar

Interfaces de salida:

- `RAM_outData`: dato a leer

Funcionamiento y simulación

Una vez que la señal de habilitación `enable` está activada, se comprueba si la señal de escritura `RAM_write` está a nivel alto. Si es así, se almacena el valor en la posición de memoria que indica `RAM_address`. Por el contrario, si la señal que está activada es la de lectura, se devolverá el valor guardado en la posición de memoria que indica `RAM_address`.

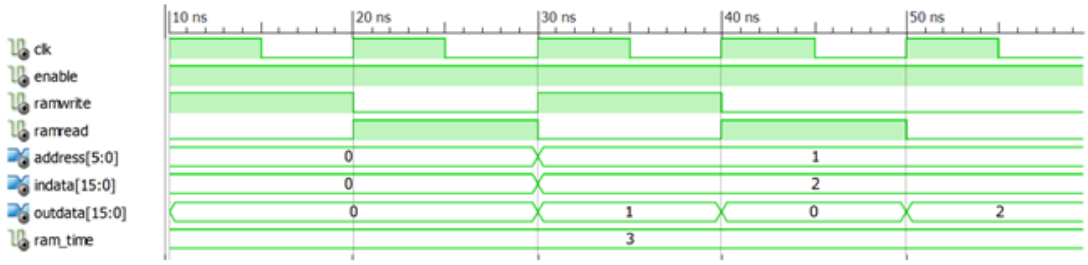


Figura 2.7: Simulación del bloque *RAM_Pilotos*

Un ejemplo de su funcionamiento se puede apreciar en la simulación de la figura 2.7.

Para comprobar el correcto funcionamiento del bloque, la simulación se ha realizado con un espaciado temporal de 4 y uno frecuencial de 8, es decir, siempre y cuando la señal de entrada `ram_time` sea 3 y la dirección de memoria sea múltiplo de 8, se insertará un piloto y, por tanto, el valor almacenado en dicha posición será igual a `1`.

En la simulación, se comienza escribiendo el valor 0 en la posición de memoria 0. Tras la escritura, se pasa a la lectura de dicho dato comprobando que, efectivamente, el valor guardado no es 0 sino 1. En cambio, si realizamos las mismas operaciones para la dirección de memoria 1, vemos que el valor almacenado es igual al valor de entrada.

2.4. xFFT

Diseño

Para la realización de la FFT/IFFT se ha optado por utilizar un *IP Core* de *Xilinx*. Los algoritmos empleados y los criterios de diseño de este *Core* se explican con más detalle en [2]. A continuación se describen brevemente los criterios más significativos.

El primer criterio de diseño a tener en cuenta es el tamaño de la xFFT. Este valor no es arbitrario. De hecho, el *IP Core* FFT está diseñado para unos tamaños $N = 2^m$ donde $m = 3$–16. De igual manera, los datos de entrada al *Core* deben tener una longitud comprendida entre 8–34.

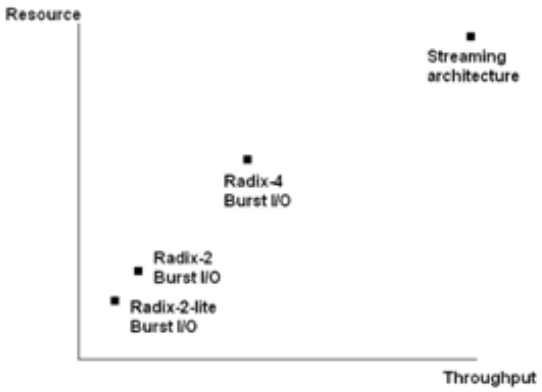


Figura 2.8: Algoritmos para el cálculo de la FFT frente al tamaño de la FFT [2]

En cuanto al algoritmo empleado, al tratarse de un FFT de 64 puntos, la mejor opción es la de utilizar el algoritmo *Radix-4* pues el número de operaciones es menor que en *Radix-2*, consiguiendo así un cálculo de la DFT mucho más rápido y eficiente. La figura 2.8 obtenida de [2] muestra lo anteriormente explicado.

A diferencia de [3] y [4], los datos obtenidos son escalados con el fin de lograr un mejor uso de los recursos de la FPGA haciendo, por tanto, que la salida del módulo tenga también una longitud de 16 bits.

A pesar de que la entrada al *Core* se hace en orden natural, en la salida se produce *'digit reversal'* haciendo que los resultados no aparezcan en orden. Sin embargo, se puede evitar este efecto haciendo que los resultados aparezcan en orden natural a cambio de un retardo en el cálculo de la FFT.

Con todas las características anteriormente descritas y utilizando la herramienta *Core Generator* de *Xilinx*, se generará el *IP core* deseado.

Finalmente, el diseño de bloques queda como el de la figura 2.9:

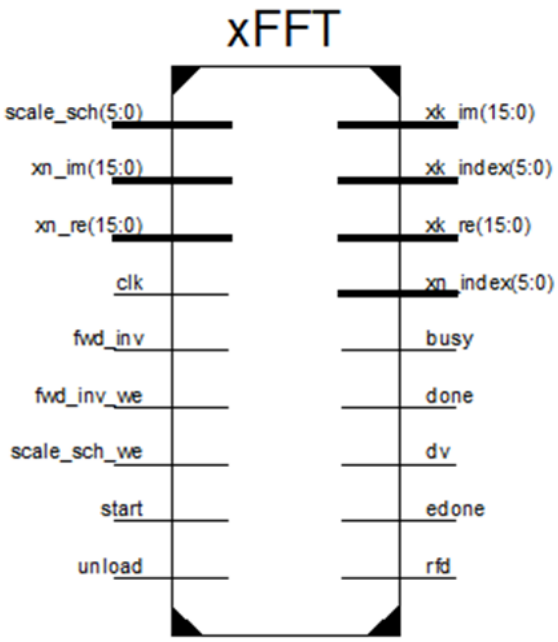


Figura 2.9: Diagrama de bloques de FFT

Interfaces de entrada:

- `clk`: reloj del sistema
- `start`: señal asíncrona para el inicio de la carga de de datos al módulo
- `unload`: inicio de la descarga
- `fwd_inv`: '1', FFT. '0' IFFT
- `fwd_inv_we`: write enable de `fwd_inv`
- `scale_sch_we`: write enable de `scale`
- `xn_re/xn_im`: entrada de datos
- `scale_sch`: tipo de escalado

Interfaces de salida:

- `rfd`: Ready for data. Activo durante la carga de datos
- `busy`: activa durante el cálculo
- `done`: se activa durante un ciclo una vez que se ha finalizado con el cálculo
- `edone`: se activa un ciclo después de `done`
- `dv`: data valid. Activo durante la descarga.
- `xn_index/xk_index`: índices de los datos de entrada/salida
- `xk_re/xk_im`: salida de datos reales e imaginarios

Funcionamiento y simulación

Aunque su funcionamiento se explica con más detalle en [2], a continuación se ilustra brevemente su funcionamiento. Durante un ciclo de reloj las señales `start` y `fwd_inv_we` deben permanecer a nivel alto para la iniciación de la carga de datos y la configuración de la FFT inversa o directa como se puede ver en la figura 2.10.

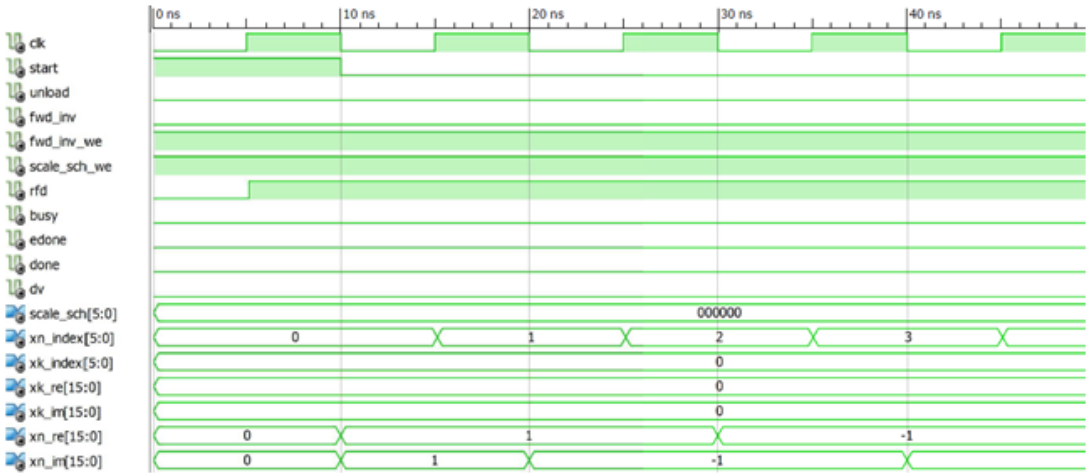


Figura 2.10: Simulación del inicio y carga de datos al módulo FFT

En cada ciclo de reloj se pone en la entrada del *core* los valores reales e imaginarios de los N símbolos. Una vez cargados todos los datos, se inicia el cálculo de la IFFT, poniéndose a nivel alto la señal `busy` como se puede comprobar en la figura 2.11.

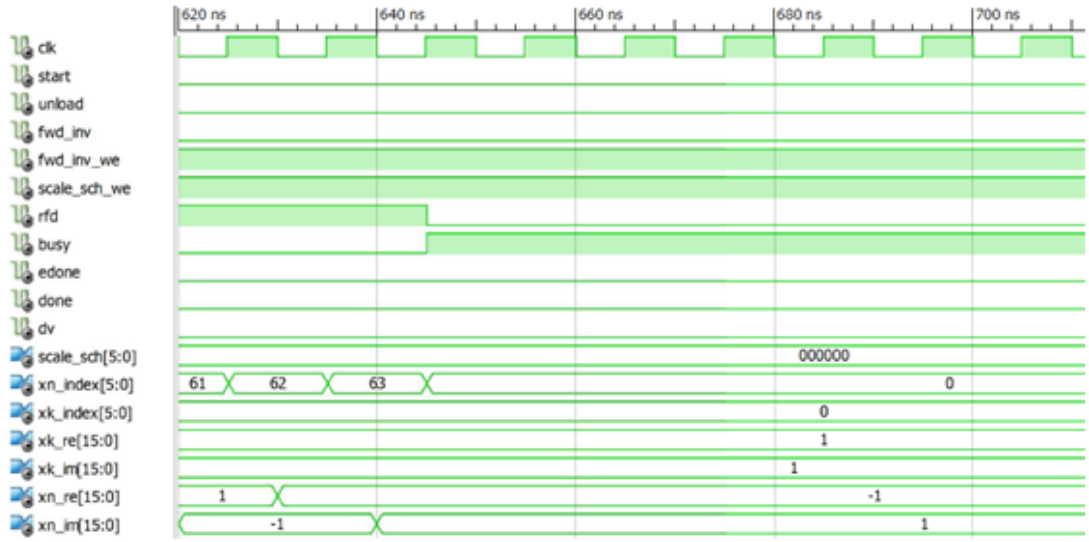


Figura 2.11: Simulación del inicio del cálculo del módulo FFT

La duración del cálculo de la FFT es un valor que proporciona el *Core Generator*. Para nuestro diseño, considerando un total de 64 portadoras, es de 253 ciclos desde el inicio de la carga de datos hasta su total descarga. Sin embargo, el valor que se desconoce es el tiempo que tarda el *Core* en obtener los valores de salida. El conocimiento de este valor es completamente necesario pues para iniciar la

descarga de datos hay que habilitar la señal `unload` del *Core*, por ello se utilizan las señales `busy` y `done` para saber que ha finalizado la operación.

Por otra parte, dado que en el diseño se ha buscado que los resultados estén en orden natural, hay que esperar 8 ciclos de reloj adicionales hasta el inicio de la descarga de datos. Al igual que en la carga, la descarga se realiza elemento a elemento, es decir, un elemento por cada ciclo de reloj.

Observando la figura 2.12 se puede comprobar lo anteriormente expuesto.

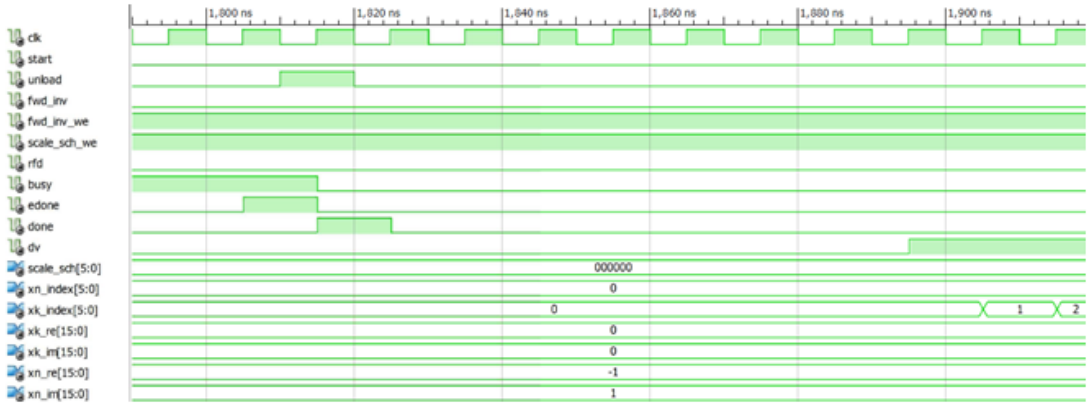


Figura 2.12: Simulación de la descarga de datos del módulo FFT

Como se ha podido observar, este bloque es el que más va a retardar la generación de la trama OFDM. Como se verá en el capítulo 6, esta latencia junto con la del resto del sistema provocará la aparición de un tiempo de procesamiento bastante extenso, incluso mayor que el tiempo de transmisión.

2.5. RAM_OFDM

Diseño

Al igual que en *RAM_Pilotos* (sección 2.3), se trata de una memoria RAM síncrona. La función de este bloque del sistema modulador es la de almacenar los resultados del módulo *xFFT*. Suponiendo una trama de N_OFDM símbolos con un total de N portadoras, el tamaño de la memoria tiene que ser, por tanto, de $N\_OFDM \cdot N$ posiciones, es decir, de 6400 posiciones según los valores escogidos en el diseño.

El diagrama de bloques de *RAM_OFDM* se muestra en la figura 2.13.

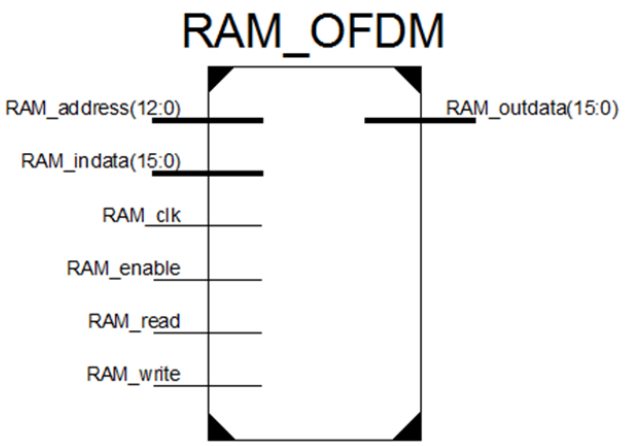


Figura 2.13: Diagrama de bloques de la memoria *RAM_OFDM*

Interfaces de entrada:

- `RAM_clk`: señal de reloj del sistema
- `RAM_enable`: señal de activación
- `RAM_write`: habilitar escritura
- `RAM_read`: habilitar lectura
- `RAM_address`: posición de memoria para lectura/escritura
- `RAM_inData`: dato a guardar

Interfaces de salida:

- `RAM_outData`: dato a leer

Funcionamiento y simulación

Su funcionamiento es el mismo que el explicado anteriormente para la *RAM_pilotos*, es decir, siempre que la señal de habilitación de la memoria esté activa, se comprobará si están activas las señales de lectura o escritura. En caso afirmativo, se devolverá o se guardará el valor de la posición de memoria indicada por `RAM_address`.

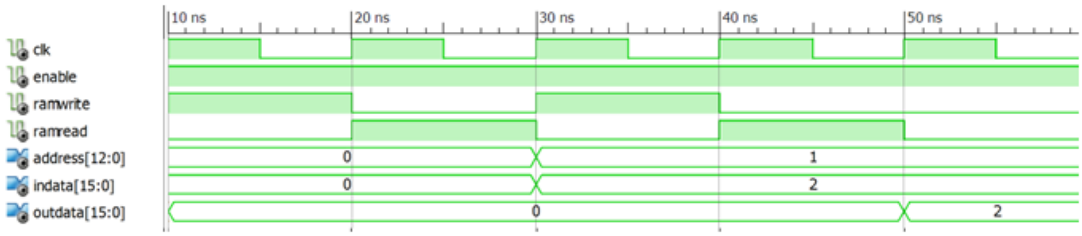


Figura 2.14: Simulación de la memoria *RAM_OFDM*

La simulación de la figura 2.14 muestra el funcionamiento de esta memoria: una vez activada la señal de habilitación `RAM_enable`, se puede escribir y leer, en este caso, de las posiciones 0 y 1 de memoria.

2.6. Modulador

El modulador es el bloque clave en el diseño de nuestro transmisor OFDM. En el esquema de la figura 2.2, se puede ver cómo el modulador se encuentra en un nivel jerárquico superior, haciendo que los bloques anteriormente descritos formen parte de él, de ahí que su función principal sea la de realizar el control de dichos bloques.

Su diseño, aunque a primera vista parezca que no haya sufrido ninguna modificación con respecto a las anteriores fases del proyecto, esos cambios han sido vitales no sólo en el diseño sino también en la síntesis y en su posterior implementación.

Diseño

Con lo anteriormente descrito, ya se puede diseñar el bloque *Modulador*. En cuanto a sus entradas, además de las señales de reloj, de reset y de habilitación, son necesarias una entrada para escoger el tipo de modulación y otra para los bits de información generados por el bloque transmisor. Por otra parte, como salida tendrá los símbolos reales e imaginarios junto con una señal que indicará si hay datos válidos en esas salidas, además de una señal que informa al transmisor que el modulador está listo para recibir bits de información.

Su diseño de bloques se puede observar en la figura 2.15.

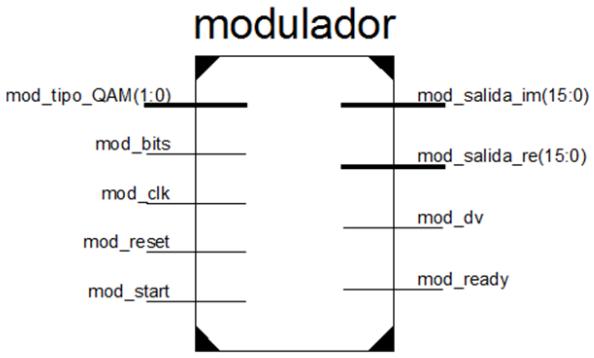


Figura 2.15: Diagrama de bloques del *Modulador*

Interfaces de entrada:

- `mod_clk`: reloj del sistema
- `mod_reset`: señal asíncrona
- `mod_bits`: bits a transmitir
- `mod_start`: señal para iniciar el proceso de generación de trama
- `mod_tipo_QAM`: 4,8,16 ó 64 QAM

Interfaces de salida:

- `mod_ready`: el modulador está listo para recibir los bits de entrada
- `mod_salida_re`: salida símbolos reales
- `mod_salida_im`: salida símbolos imaginarios
- `mod_dv`: indica que hay datos válidos a la salida

Funcionamiento y simulación

Como ya se ha explicado, el bloque modulador es el encargado de controlar y gestionar todos los bloques anteriormente comentados. Es decir, de activar las señales de habilitación de cada uno de los bloques en el momento adecuado, de extraer los datos de las memorias para pasarlas al siguiente bloque, esperar a

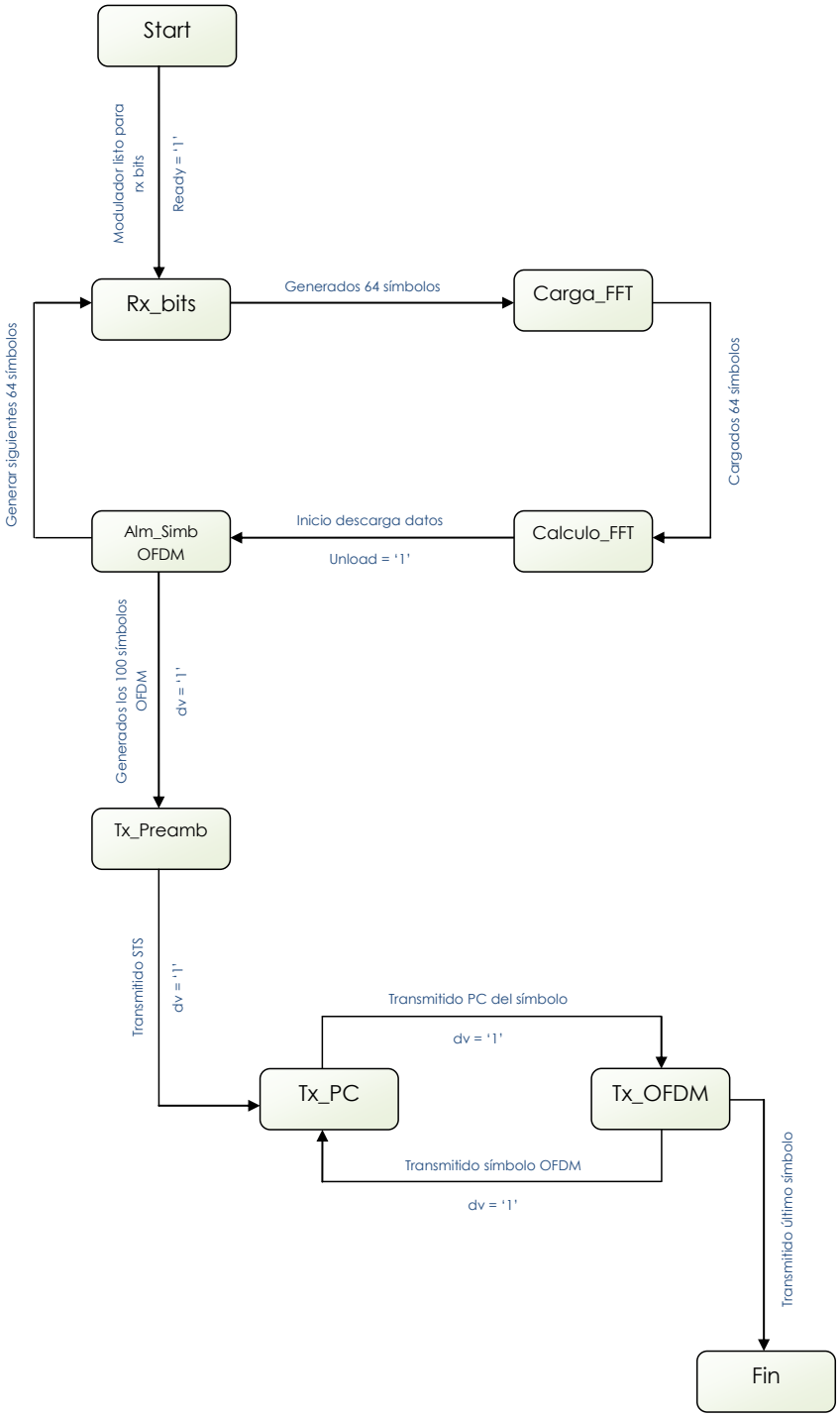


Figura 2.16: Diagrama de estados de *Modulador*

que se realice el cálculo de determinadas funciones, etc. Este control se realiza a través de la máquina de estados de la figura 2.16.

En el esquema de la figura 2.16, para hacer más fácil su comprensión, se ha preferido simplificar las condiciones que hacen pasar de un estado al siguiente. Estas condiciones se detallarán en cada uno de los estados.

Start

En este estado se inicializan todas las variables y se mantiene a la espera de la señal de activación. Tras la inicialización, se activa la señal `ready` indicando así al bloque *Transmisor* que está listo para recibir los bits de entrada. Una vez que el bloque *Transmisor* observa que el *Modulador* está preparado para la recepción de datos, pone a nivel alto la señal de activación del bloque `start`, iniciando así el proceso de generación de la trama.

Rx_bits

Este estado controla tanto la conversión bits-símbolos como su almacenado y la inserción de las señales de referencia.

En primer lugar, se genera el símbolo 4-QAM, para lo cual es necesario que el bloque *Transmisor* envíe dos bits. Una vez que el bloque *Bits2Simb* ha realizado la conversión, activa la señal `b2s_out` indicando así a la memoria que el símbolo está listo para almacenar.

A continuación, antes de guardar el dato, la memoria *RAM_Pilotos* comprueba si la posición del símbolo se corresponde con la ubicación de un piloto. En caso afirmativo, el símbolo tomará el valor asignado a las señales de referencia, que en el sistema implementado se corresponde con un 1. Si por el contrario, no se corresponde con un piloto, el símbolo almacenado no sufrirá ninguna modificación.

La figura 2.17 muestra el funcionamiento en simulación. Como se puede ver, los símbolos se almacenan cada dos ciclos en la posición de memoria indicada por `mod_ram_address` siempre y cuando la señal de activación de lectura `mod_ram_write` esté a nivel alto. Generados los 64 símbolos, se produce el cambio de estado a `carga_fft`.

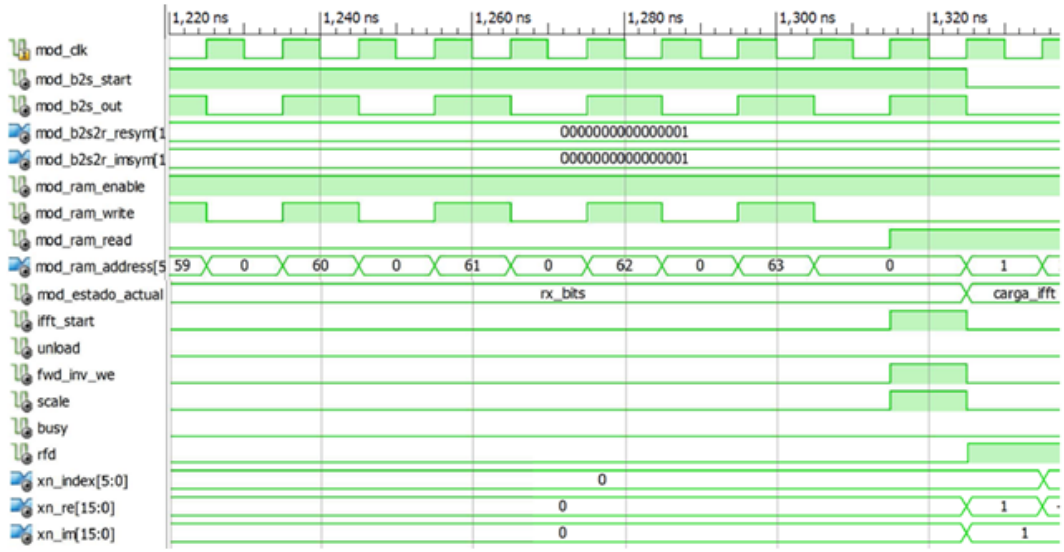


Figura 2.17: Simulación de la generación y almacenamiento de los símbolos generados

Carga_FFT

Se encarga de cargar en el módulo FFT los datos de las dos memorias *RAM_pilotos*. Como ya se explicó en la sección 2.4, para iniciar el cálculo de la IFFT hay que activar las señales de `start` para iniciar el cálculo, `fwd_inv_we` para que el módulo lea la señal `fwd_inv` y así conocer si se quiere realizar la FFT directa o inversa y la señal `scale_sch` para escalar los datos de salida.

Durante el proceso de carga, la señal `rfd` (Ready For Data) permanecerá activa. Tras la carga, `rfd` volverá a nivel bajo, siendo `busy` la que esté activa durante el cálculo.

En la figura 2.17, podemos ver todo este proceso de habilitación y deshabilitación de entradas.

Calculo_FFT

Se permanecerá en este estado hasta que la señal `busy` deje de estar activa y la señal `edone` pase a estar a nivel alto. En ese momento, se activará la descarga de datos mediante la señal `unload`.

En la figura 2.18, se puede comprobar el cambio de estado entre `Calculando_FFT` y `almacena_simb_OFDM`, con todo el cambio de señales que conlleva.

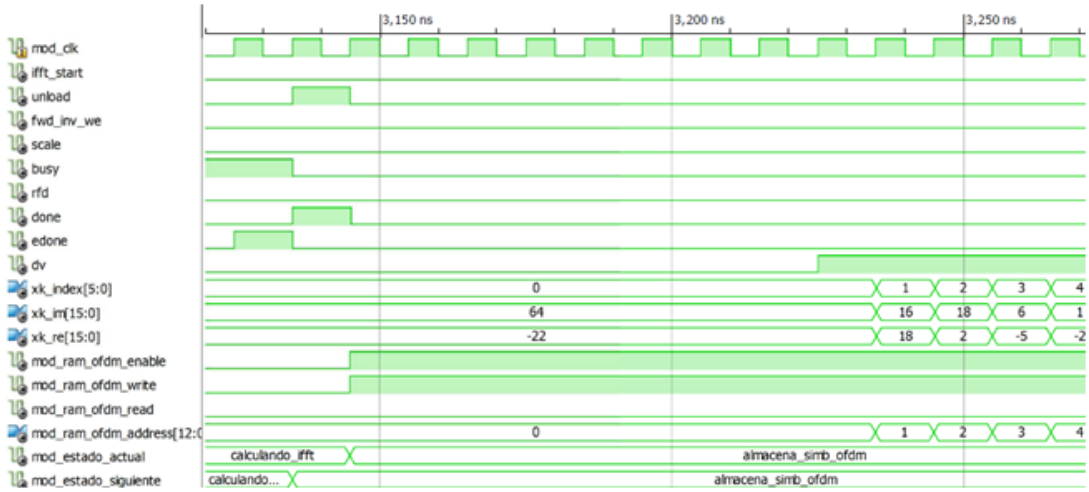


Figura 2.18: Simulación del paso del estado `Calculando_OFDM` a `Almacena_Simb_OFDM`

Almacena_Simb_OFDM

Se encarga de almacenar los N_OFDM símbolos OFDM de la trama. En primer lugar, una vez esperados los 8 ciclos antes de la descarga de datos del bloque $xFFT$, se almacenan los N valores del símbolo OFDM. Durante todo este proceso de descarga, la señal `dv` del bloque $xFFT$ estará activada, indicando que hay datos en la salida.

En la simulación de la figura 2.18, se puede verificar que efectivamente hay un periodo de espera para el inicio de la descarga de 8 ciclos de reloj, tras el cual, la señal `dv` queda habilitada, comenzando la descarga de datos y, por tanto, su almacenamiento en las memorias RAM correspondientes.

En la simulación también se puede ver cómo la dirección de memoria de las RAM es incrementada con el índice de los datos de descarga `xk_index` y, dado que la señal de habilitación de escritura está activada y junto con lo anteriormente explicado en la sección 2.5, se puede decir que los datos están siendo guardados correctamente.

Si ya tenemos los $N \cdot N\_OFDM$ símbolos se iniciará la transmisión de la trama como ocurre en la simulación de la figura 2.20, en la cual se ve cómo se almacena la última muestra del último símbolo OFDM, cambiándose así de estado. En caso contrario, se volverá al estado `Rx_bits` para generar los siguientes N valores que forman parte del símbolo OFDM, como ocurre en la simulación de la figura 2.19.

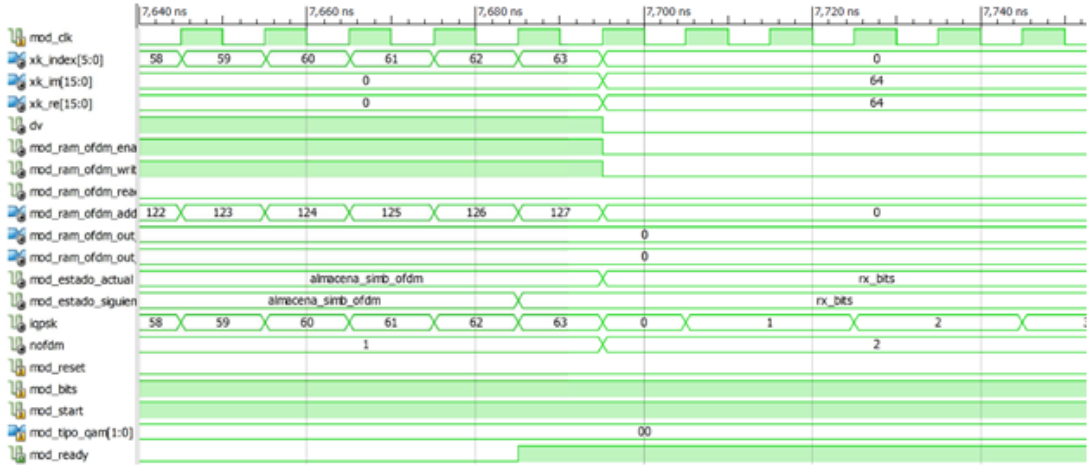


Figura 2.19: Descarga y almacenamiento de los símbolos OFDM

Transmitiendo_Preambulos

Como ya se ha explicado en capítulos anteriores, el STS está formado por la repetición de 16 símbolos. Por tanto, este estado se encarga de enviar 10 veces los símbolos cortos de entrenamiento.

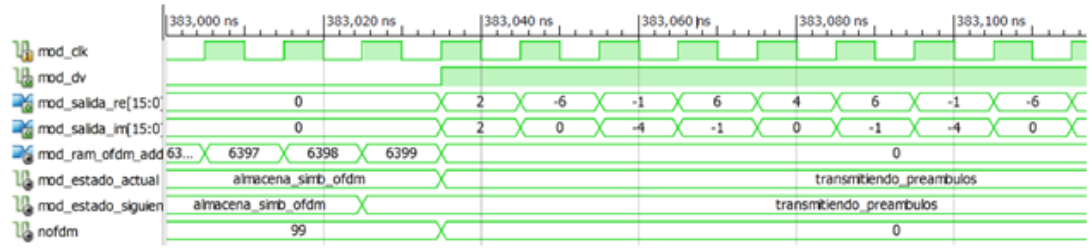


Figura 2.20: Inicio de la transmisión de datos

Por primera vez, se activa la salida `dv` del modulador indicando, por tanto, el inicio de la transmisión.

Transmitiendo_PC

Tras los preámbulos se comienza con la transmisión de los símbolos OFDM generados junto con el prefijo cíclico asociado a cada símbolo. Este estado es el que se ocupa de tomar las 8 últimas muestras del símbolo OFDM y transmitirlas antes

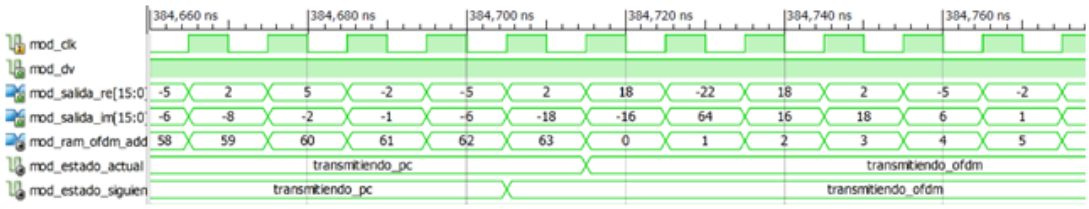


Figura 2.21: Transmisión del PC y del símbolo OFDM

del mismo tal y como se puede comprobar en la simulación de la figura 2.21 a través de la señal `mod_ram_ofdm_address`.

Transmitiendo_OFDM

Por último, se transmite el símbolo OFDM. Si se trata del último símbolo se volverá al estado `Start` a la espera de generar una nueva trama, pasando previamente por el estado `Fin`, en cual se inicializarán las variables. Por el contrario, si no es el último símbolo, se volverá al estado `Transmitiendo_PC` con el fin de transmitir el PC asociado a dicho símbolo antes de enviarlo.

2.7. Transmisor

Por último, queda por explicar el bloque *Transmisor*. Como se ha podido ver en la figura 2.2, el *Transmisor* es el bloque de mayor jerarquía del diseño. Su función es la de generación del flujo de bits que le llega al *Modulador* para que éste los procese y sea capaz de generar los símbolos que formarán parte de la trama.

Diseño

Su diseño, a diferencia del resto de bloques, no ha sufrido prácticamente cambios con respecto a fases anteriores del proyecto. Sólo consta de dos entradas: la señal de reloj del sistema y la señal de reset. Por otra parte, como salidas tiene los símbolos reales e imaginarios y la señal que indica que hay datos válidos a la salida.

La figura 2.22 muestra como quedaría el diagrama de bloques del *Transmisor*.

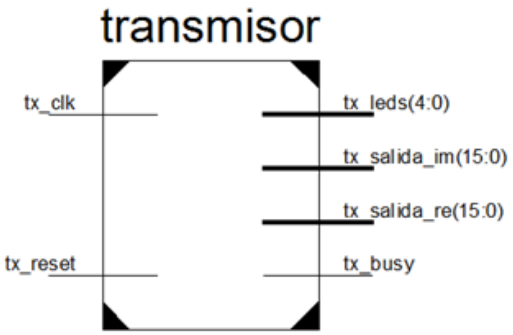


Figura 2.22: Diagrama de bloques del *Transmisor*

Interfaces de entrada:

- `clk`: reloj del sistema
- `reset`: señal asíncrona

Interfaces de salida:

- `tx_busy`: indica que hay datos a la salida
- `tx_salida_re`: salida símbolos reales
- `tx_salida_im`: salida símbolos imaginarios

Funcionamiento y simulación

Al igual que el bloque *Modulador*, el funcionamiento está basado en una máquina de estados. Dado que la única función es la generación de bits, la máquina de estados es bastante más sencilla, constando solamente de tres estados. En la figura 2.23, se puede ver su diagrama de estados simplificado.

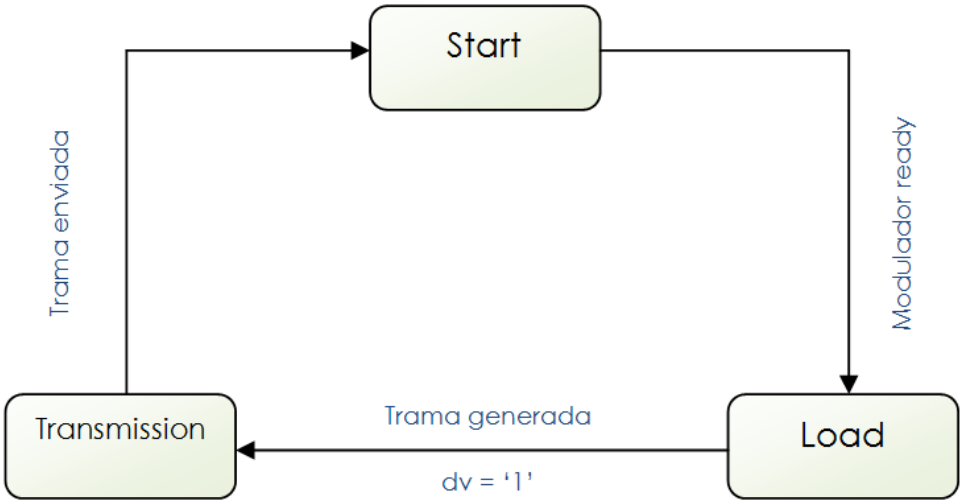


Figura 2.23: Diagrama de estados del bloque *Transmisor*

Start

En este estado, además de inicializarse las variables, se inicia la generación del flujo de bits. Para ello, permanecerá a la espera a que el *Modulador*, a través de su salida `ready`, indique que está preparado para recibir datos. Una vez activada la señal `ready`, el *Transmisor* pondrá nivel alto la señal `start`, iniciando así el proceso de generación de trama.

En la simulación de la figura 2.24 se puede comprobar lo anteriormente explicado.

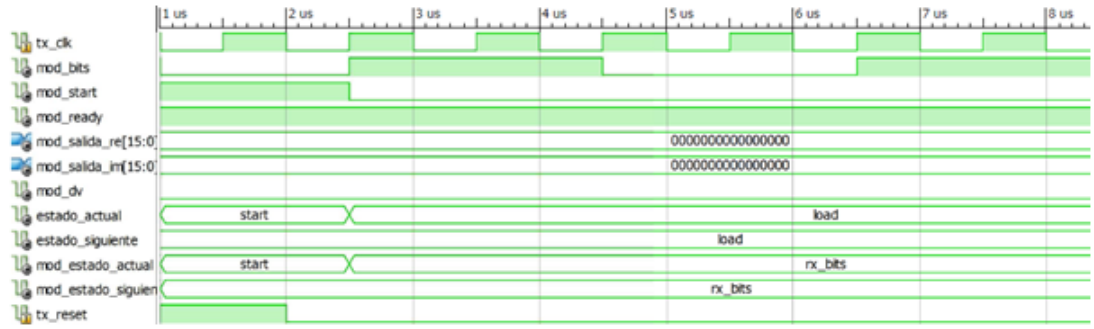


Figura 2.24: Simulación del inicio del bloque *Transmisor*

Load

Se irán enviado los bits al *Modulador* hasta que este haya procesado todos los datos y tenga la trama generada. En ese momento, el *Modulador* activará la salida `dv` iniciando así el proceso de transmisión.

En la figura 2.25, se puede ver cómo el cambio de estado se produce en el momento que el *Modulador* ha terminado de originar la trama.

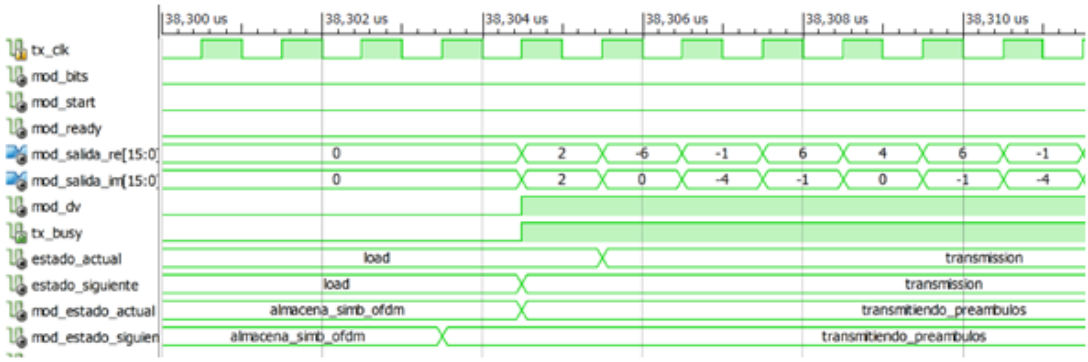


Figura 2.25: Simulación de la carga de datos al bloque *Modulador*

Transmission

Este estado se limita a poner en la salida los símbolos que el *Modulador* ha generado, activando la señal de ocupado `busy`. Una vez transmitidos los N_OFDM símbolos OFDM, volverá de nuevo al estado `Start`, permaneciendo así a la espera de originar una nueva trama OFDM.

En la simulación de la figura 2.26, se puede comprobar cómo una vez transmitido el último símbolo se vuelve al estado inicial.

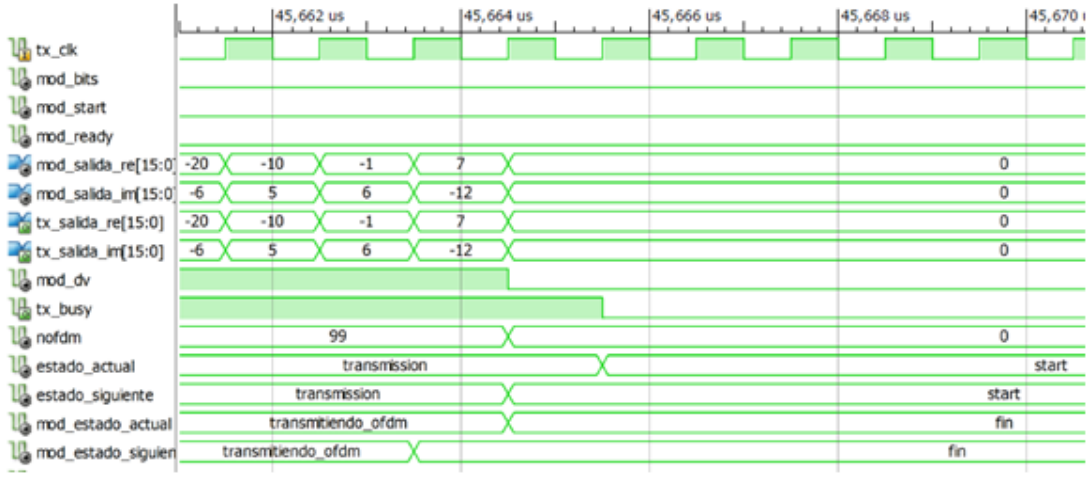


Figura 2.26: Simulación de la transmisión de datos

Una vez explicados cada uno de los bloques del sistema tan sólo quedaría mostrar el conexionado de bloques, el cual se muestra en las figuras 2.27 y 2.28.

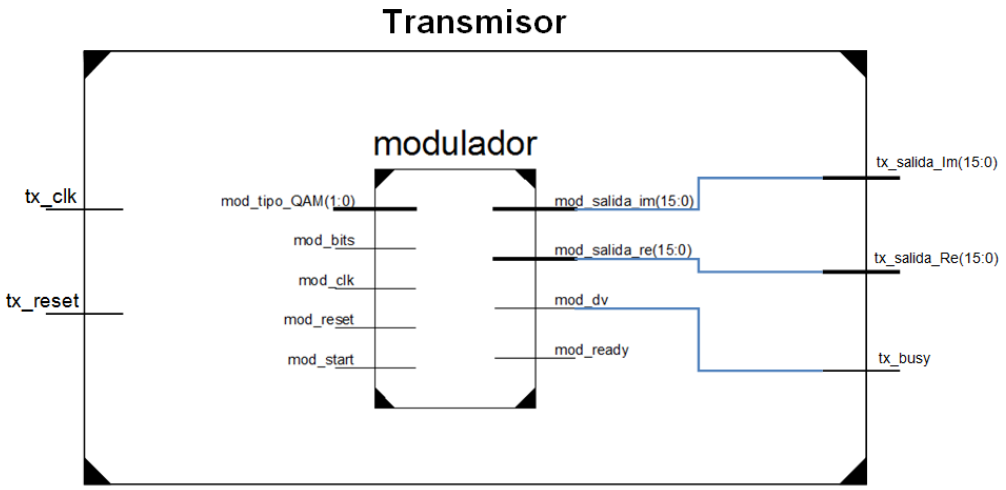


Figura 2.27: Diagrama de bloques del *Transmisor*

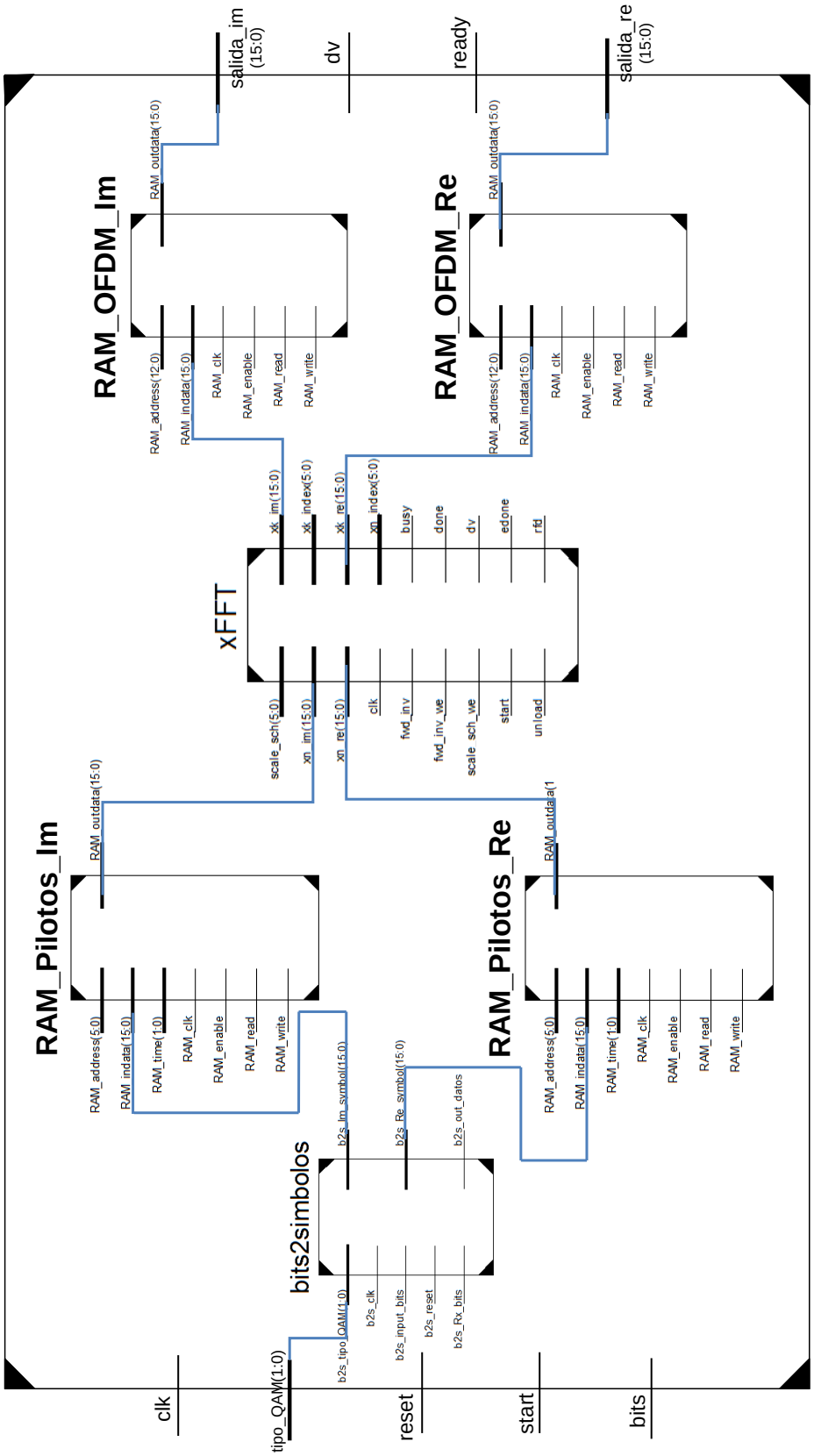


Figura 2.28: Diagrama de bloques del *Modulador*

Capítulo 3

Receptor

En este capítulo se presenta el diseño e implementación del receptor OFDM, detallando al igual que se hizo con el transmisor, cada uno de los bloques que forman parte de él.

Como se irá viendo a lo largo de esta sección, las operaciones realizadas por el receptor resultan bastante críticas, especialmente el sincronismo gracias al cual se podrá demodular y conseguir así los bits enviados por el transmisor.

Por último, una vez implementado en VHDL el diseño, se mostrarán las simulaciones de dichos bloques pudiendo comprobar así su correcto funcionamiento.

3.1. Descripción del sistema

Como ya se ha explicado anteriormente en la sección 1.2, la función del receptor es la de conseguir los bits de información enviados por el transmisor. Sin embargo, hay que tener un especial cuidado con el diseño del mismo, principalmente por el tiempo de procesamiento, el cual desempeña un papel fundamental en la implementación del receptor como se irá viendo a lo largo de esta sección.

Como ya ocurrió en el transmisor, el esquema del receptor, mostrado de nuevo en la figura 3.1, no se corresponde exactamente con el diseño implementado.

En primer lugar, como se verá a continuación, se necesitan memorias para ir almacenando los datos que se van recibiendo.

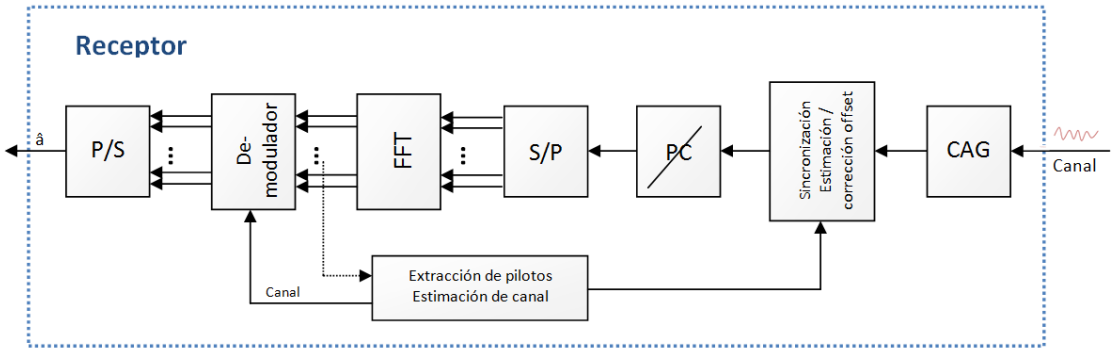


Figura 3.1: Esquema del bloque *Receptor*

En cuanto al Control Automático de Ganancia (AGC), no estará implementado en esta fase del diseño aunque sí consta de un amplificador tras la conversión analógico-digital.

De igual forma, la estimación y la corrección del offset quedan pendientes también para próximas fases, contando nuestro diseño con un ecualizador gracias al cual se conseguirá reducir el número de errores.

Por último queda mencionar que los bloques Serie/Paralelo y Paralelo/Serie no son necesarios por las mismas razones expuestas en la sección 2.1.

Finalmente, el esquema del *Receptor* implementado se muestra en la figura 3.2. Como se puede observar, el receptor consta de 3 bloques: un demodulador en cuadratura, una memoria RAM y el demodulador.

El demodulador en cuadratura, realizado en fases anteriores del proyecto [3], es el encargado de separar la señal procedente del ADC en dos señales, una en fase y otra en cuadratura.

Toda la información que es recibida debe ser almacenada en el sistema para su procesamiento, de ahí, que tras la demodulación IQ sea totalmente necesario la implementación de memorias RAM para guardar todos los datos que le van llegando.

Una vez que se tiene toda la información almacenada ya se puede iniciar el proceso de sincronismo con el fin de poder demodular la trama de datos, consiguiendo así los bits enviados por el transmisor.

Con todo esto, se puede decir que el *Receptor* realiza dos operaciones en paralelo. La primera, anteriormente descrita, es la encargada de guardar en la memoria

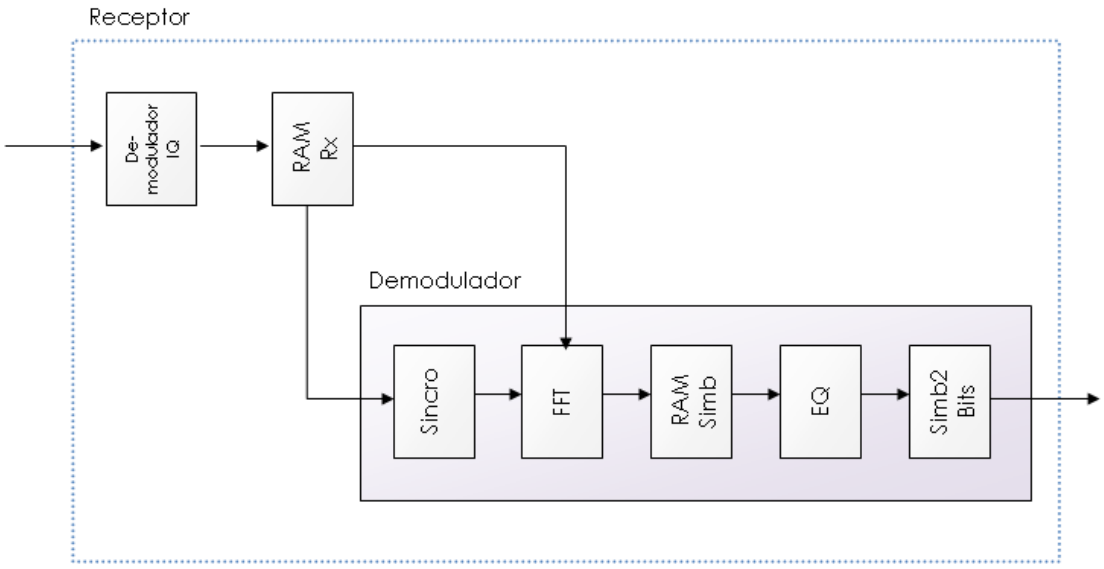


Figura 3.2: Esquema del bloque receptor implementado

RAM la información recibida. La segunda, en cambio, realiza la sincronización y demodulación del sistema. A continuación se explicarán cada una de estas operaciones.

Antes de explicar cada uno de los bloques que componen el *Receptor*, la figura 3.3 muestra las interfaces de entrada y salida del receptor diseñado.

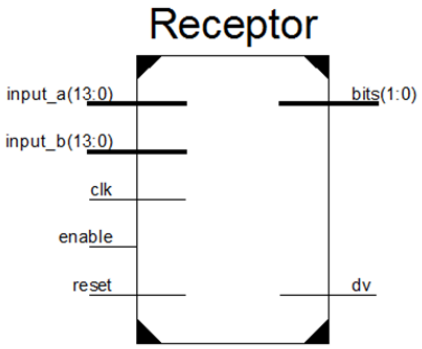


Figura 3.3: Diagrama de bloques del *Receptor*

Interfaces de entrada:

- `clk`: señal del reloj
- `reset`: señal asíncrona de reinicio
- `input_a/input_b`: entrada de datos señal en fase/señal en cuadratura

Interfaces de salida:

- `dv`: datos válidos a la salida
- `bits`: salida de datos

Resumiendo, el *Receptor* se limita a gestionar todos los bloques que lo contienen. Basicamente controla las memorias RAM, haciendo que los datos que van llegando sean guardados además de hacer que estos lleguen al *Demodulador* en cuanto este los solicita.

3.2. RAM_Rx

Diseño

Este bloque es el encargado de almacenar toda la información recibida en el *Receptor*. Aunque aparentemente esta memoria no debe ser muy diferente a las anteriormente explicadas en la sección 2.1, su diseño resultará mucho más complejo.

La primera cuestión que surge es el tamaño de las memorias pues, como se ha ido haciendo en el *Transmisor*, habrá dos memorias, una para la parte real de los datos modulados y otra para la parte imaginaria. Teniendo en cuenta que la trama transmitida consta de 100 símbolos OFDM con 64 portadoras, que generan un total de 6400 muestras a los cuales hay que sumar las 160 muestras procedentes de los preámbulos y las 800 de los prefijos cíclicos. Es decir, se necesitará una memoria de un tamaño mayor a la longitud total de nuestra trama, que es de 7360 muestras.

Por otra parte, el tiempo de procesamiento es el que nos va a marcar cuánto espacio de memoria adicional se necesita además del tamaño de la trama. Para

ello, se necesita conocer el tiempo que tarda el *Receptor* desde el momento en el que se recibe el primer símbolo de los preámbulos hasta que se termina la demodulación y se tengan los bits recibidos. Con todos estos datos y tras la realización de varias simulaciones que se pueden ver a lo largo de este capítulo, el tamaño final de la memoria es de 65536 posiciones.

Otro aspecto de gran interés en el diseño de la memoria del *Receptor* es que dicha memoria debe permitir tanto la lectura como la escritura en un mismo ciclo de reloj, sin que además se produzca ningún tipo de colisión.

Finalmente, como se ha hecho anteriormente, debido a un mejor aprovechamiento de los recursos de la FPGA se ha preferido el uso de *IP Cores*. El *Core* utilizado es el de memoria bloque de tipo *Simple Dual Port RAM*, permitiéndonos la lectura y escritura simultánea. En cuanto al tamaño, cada espacio de memoria tendrá una longitud de 16 bits y el tamaño total, como se ha dicho antes, será de 65536, para aprovechar todas las posibles direcciones de memoria en los bits.

En la figura 3.4 se muestra su diseño de bloques.

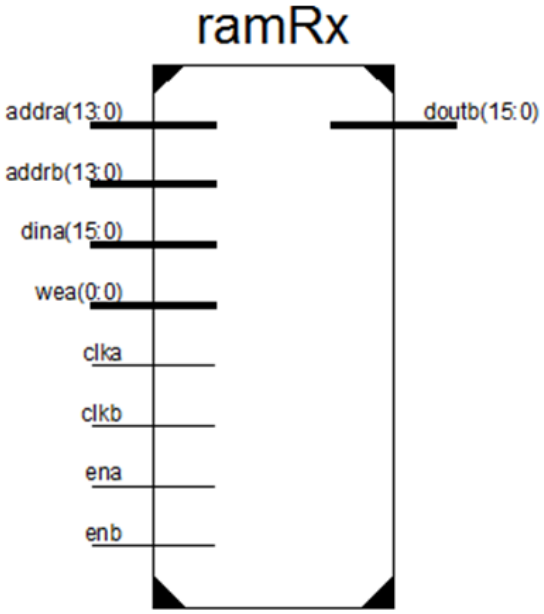


Figura 3.4: Diagrama de bloques de la memoria RAM del *Receptor*

Interfaces de entrada:

- `clka`: reloj del puerto A o de escritura
- `ena`: señal de habilitación del puerto A o de escritura
- `wea`: señal de habilitación escritura
- `addra`: dirección de memoria del puerto A o de escritura
- `dina`: datos a guardar
- `clkb`: reloj del puerto B o del lectura
- `enb`: señal de habilitación del puerto B o de lectura
- `addrb`: dirección de memoria del puerto B o de lectura

Interfaces de salida:

- `doutb`: datos de salida

Funcionamiento y simulación

Su funcionamiento es igual a cualquier memoria RAM, con la única peculiaridad que se puede leer y escribir al mismo tiempo.

Para realizar la escritura, se activa la entrada `ena` y se pone en `dina` el dato que se quiere guardar en la posición de memoria indicada por `addra`.

En cambio, para la lectura, además de habilitar la lectura a través de la entrada `enb`, con la interfaz `addrb` se indica la posición que se desea leer.

En la simulación de la figura 3.5, se puede observar el correcto funcionamiento de este bloque con un ejemplo.

En primer lugar, se escribe en la posición `1` el valor `4`. En el siguiente ciclo, se escribe el valor `6` en la posición `2` a la vez que se lee la posición `1`. Como bien se explica en [6], hay un retardo de 3 ciclos desde que se indica la posición de lectura hasta que se obtiene el resultado en la salida. En la simulación, se ve que efectivamente existe esta latencia, algo que habrá que tener en cuenta en la lectura de datos como se irá viendo a lo largo de esta sección.

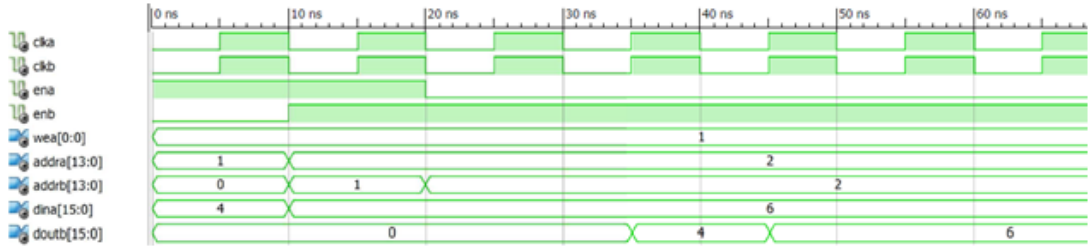


Figura 3.5: Simulación de la memoria RAM del *Receptor*

3.3. Demodulador

Diseño

El esquema del *Demulador* implementado se puede ver en la figura 3.2. Como se ve consta de 4 bloques: un módulo de sincronización, uno de FFT, una memoria RAM y el bloque *Simb2Bits*. Por tanto, la función del bloque *Demodulador* es la de gestionar todos los módulos anteriormente mencionados.

Como se puede observar en el esquema de la figura 3.2, la finalidad de este bloque es la de convertir los símbolos OFDM almacenados en la memoria RAM del *Receptor* en bits realizando, como ya se ha ido explicando, las operaciones inversas a las ejecutadas por el bloque *Transmisor*.

En primer lugar, se realiza la sincronización en la que, como se explicará en el capítulo 4, sólo se desarrolla la sincronización temporal con el fin de encontrar el inicio de la trama de datos.

Una vez conseguida la sincronización, se debe eliminar el prefijo cíclico antes de iniciar la carga de datos en el módulo FFT. Esta acción, realizada por el *Demodulador*, resulta bastante sencilla pues sólo hay que obviar dichos datos y cargar en la FFT los datos útiles de la trama.

El resultado de la FFT es guardado en una memoria RAM, pues hasta que no se tengan todos los símbolos de la trama en el dominio frecuencial no se podrá proceder a la ecualización.

Por tanto, la eliminación del PC y el cálculo de la FFT se hará tantas veces como símbolos OFDM tenga la trama recibida. En este caso, la trama consta de 100 símbolos.

Una vez obtenidos todos los símbolos, se ecualizan y, por último, se realiza la conversión símbolo-bits usando, como ya se hizo en el *Transmisor*, una constelación 4-QAM como la de la figura 2.3.

El diagrama de bloques del *Demodulador* queda como el de la figura 3.6.

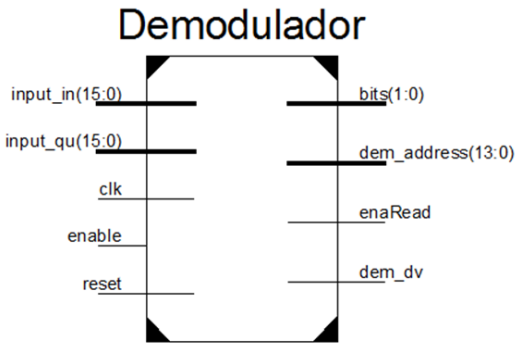


Figura 3.6: Diagrama de bloques del *Demodulador*

Interfaces de entrada:

- `clk`: reloj del sistema
- `reset`: señal asíncrona de inicialización
- `enable`: señal asíncrona de habilitación
- `input_in`: entrada de datos en fase
- `input_qu`: entrada de datos en cuadratura

Interfaces de salida:

- `dem_dv`: salida para indicar que hay datos válidos a la salida
- `ena_read`: señal de habilitación de lectura de la memoria *RAM_Rx*
- `dem_address`: dirección de memoria a leer de la memoria *RAM_Rx*
- `bits`: flujo de bits demodulados

Antes de pasar a explicar cómo gestiona el demodulador todas estas operaciones, se explicarán los módulos que lo componen.

La sincronización se explicará en el capítulo 4; el módulo *xFFT* ya se explicó en el *Transmisor*, en la sección 2.4. Por tanto, a continuación se detallarán tan sólo los bloques *Simb2Bits* y la memoria RAM *RAMSimb*.

3.3.1. RAM_Simb

Diseño

Esta memoria es la encargada de almacenar el resultado de la FFT de todos los símbolos OFDM recibidos.

Como ya se ha hizo en la memoria *RAM_Rx*, para un mejor uso de los recursos de la FPGA, esta memoria está generada utilizando el *IP Core Block Memory*. Sin embargo, en este caso, no es necesario la lectura y escritura simultánea, así que se usará el modo *Single Port RAM*.

El tamaño de la memoria será igual al número de símbolos enviados por el transmisor, es decir, tendrá en total 6400 posiciones de una longitud de 16 bits.

Su diseño de bloques es como el de la figura 3.7.

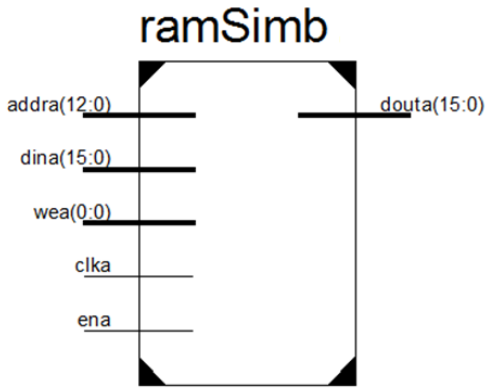


Figura 3.7: Diagrama de bloques de la memoria *RAM_Simb*

Interfaces de entrada:

- `clk`: reloj del sistema
- `ena`: señal asíncrona de habilitación
- `wea`: señal de activación de escritura
- `addra`: dirección de memoria
- `dina`: datos a almacenar

Interfaces de salida:

- `douta`: datos guardados

Funcionamiento y simulación

Su funcionamiento es como el de cualquier memoria RAM. Siempre y cuando la señal de activación `enable` esté a nivel alto, la memoria realizará sus funciones.

Para la escritura, hay que poner en `dina` el dato a guardar y en `addra` la posición de memoria en la que se quiere guardarlo, sin olvidarse de habilitar la escritura mediante la señal `wea`. Para la lectura, en cambio, la señal `wea` debe estar a nivel bajo y tan sólo se tendrá que indicar la dirección de memoria que se quiere leer. Como ya pasaba en la RAM del *Receptor*, la lectura no es inmediata y hay una espera de 3 ciclos hasta su obtención a través de la salida `douta`.

En la figura 3.8, se puede ver la simulación de la RAM. En ella se muestra tanto la lectura como la escritura de las posiciones de memoria `0` y `1`.

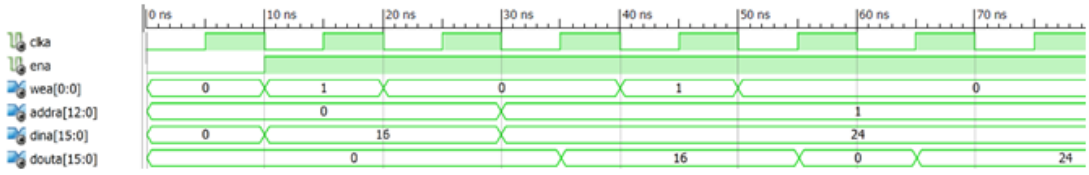


Figura 3.8: Simulación de la memoria *RAM_Simb*

3.3.2. Simb2Bits

Diseño

Este bloque es el encargado de transformar los símbolos en bits utilizando, como ya se hizo en el *Transmisor*, una constelación 4-QAM como la de la figura 2.3.

Como ya ocurrió en otros bloques del sistema, a pesar de que su diseño ya se realizó en anteriores fases del proyecto, este se ha modificado significativamente.

La primera diferencia se encuentra en el modo de operación. En la primera fase del proyecto estaba diseñado de forma asíncrona pero, dado que las memorias RAM funcionan síncronamente, hace que se dependa totalmente de la señal de reloj. La memoria *RAM_Simb* da como resultado un símbolo por cada ciclo de reloj, por tanto, en cada ciclo este bloque tomará el símbolo proporcionado por la memoria y lo convertirá en bits.

La recuperación de los bits está basada en un decisor de símbolos equiprobables. Para ello, hay que establecer unos umbrales de decisión. En este caso, al tratarse de una constelación 4-QAM equiprobable, se tomarán como umbrales el eje real e imaginario, dando lugar así a 4 posibles conjuntos de bits, mostrados en la figura 3.9.

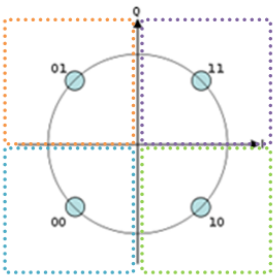


Figura 3.9: Regiones de decisión 4-QAM

El diseño consta de dos comparadores. Como los umbrales se sitúan en los ejes, la comparación se realiza, tanto para la parte real como para la imaginaria del símbolo, con 0. Es decir, siempre que la parte real o imaginaria sea mayor que 0 nos encontraremos ante un bit '1'. Por el contrario, si el resultado de la comparación es menor que 0, estaremos ante un bit '0'.

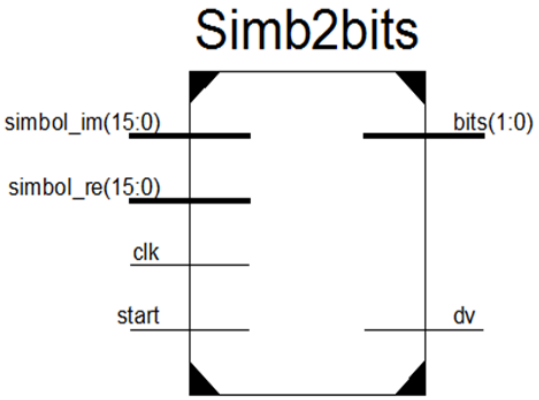


Figura 3.10: Diagrama de bloques de *Simb2Bits*

Finalmente, el diseño de bloques del conversor símbolo-bits queda como el de la figura 3.10.

Interfaces de entrada:

- `clk`: señal de reloj del sistema
- `start`: señal de activación del bloque
- `simbol_re/Simbol_im`: entrada de símbolos

Interfaces de salida:

- `bits`: salida de los bits demodulados
- `dv`: Data Valid

Funcionamiento y Simulación

Su funcionamiento prácticamente ya se ha explicado en secciones anteriores. La memoria *RAM_Simb* irá proporcionando los símbolos en cada ciclo de reloj. El decisor tomará estos símbolos convirtiéndolos en bits en función del resultado obtenido de la comparación.

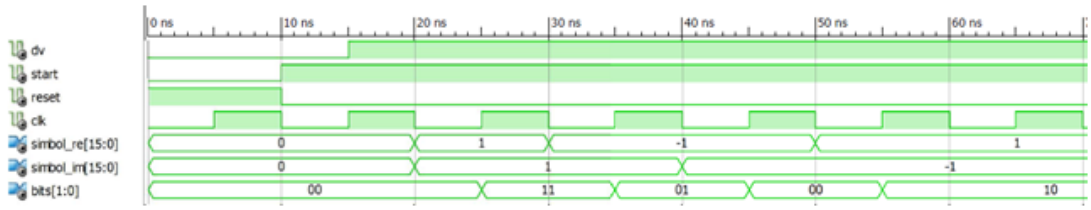


Figura 3.11: Simulación del bloque *Simb2Bits*

En la simulación de la figura 3.11 se muestra su funcionamiento para cada una de las cuatro posibles combinaciones de bits.

Una vez explicados todos los bloques que componen el *Demodulador*, se pasará a comentar su funcionamiento.

Funcionamiento y simulación

Como ya se ha dicho, la función principal del *Demodulador* es la gestionar todos los bloques que lo componen. Al igual que se hizo con otros bloques del sistema, la mejor forma de controlar todas estas operaciones es mediante una máquina de estados como la de la figura 3.12.

A continuación se explicarán uno a uno los distintos estados que forman parte de ella, dejando en color rojo aquellos bloques cuyas funciones serán explicadas en los próximos capítulos.

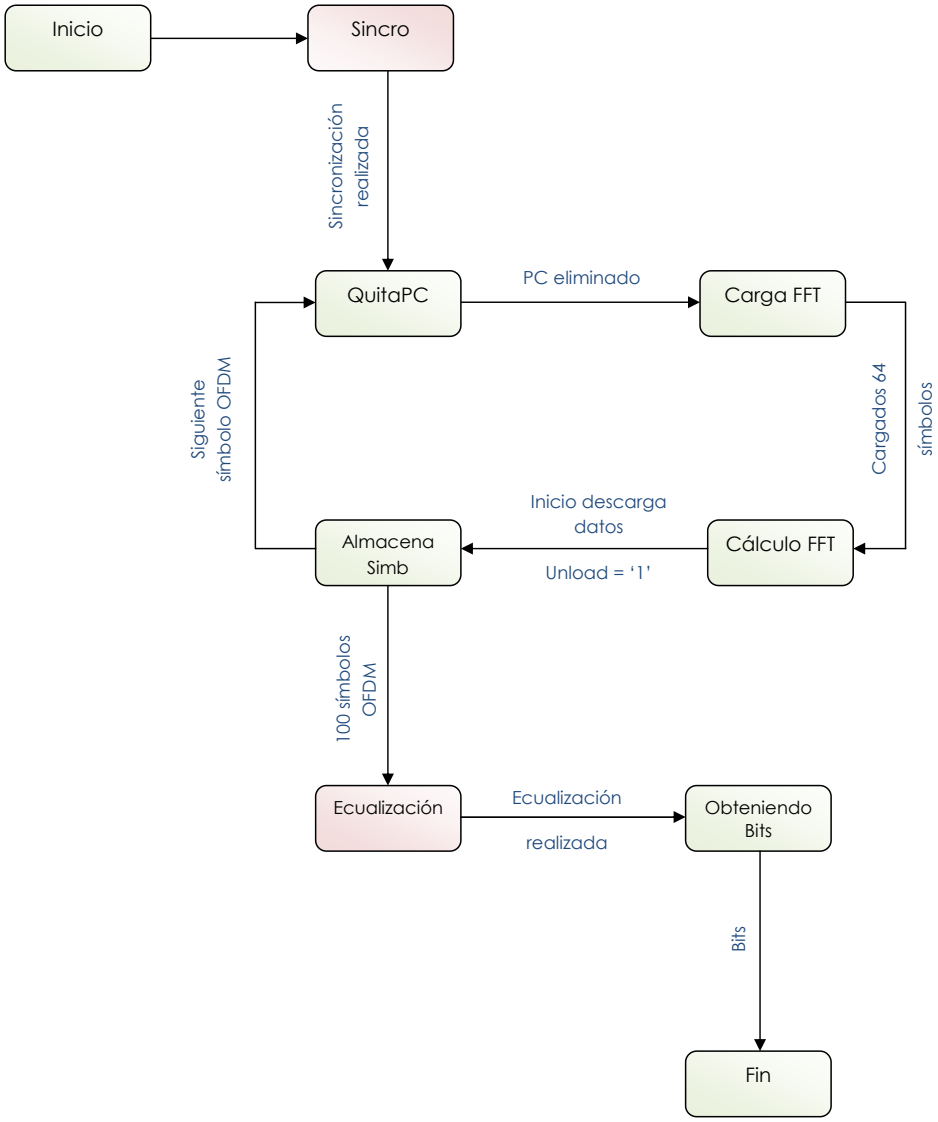


Figura 3.12: Diagrama de bloques del *Demodulador*

Inicio

En este estado se inicializan todas las variables, permaneciendo a la espera de la activación de la señal de habilitación `enable`. Una vez que se inicia el proceso de recepción de datos, habilitando la memoria RAM del receptor, se pasará a sincronizar la trama.

Sincronización

Para facilitar su compresión, todas las operaciones relativas a la búsqueda del inicio de trama, explicadas en el capítulo 4, han sido agrupadas en este bloque.

Por tanto, se permanecerá en este bloque hasta que se consiga sincronizar la trama y poder así continuar con su demodulación.

QuitaPC

Una vez realizada la sincronización, hay que quitar el PC introducido en cada símbolo OFDM. Como ya se ha explicado, dado que no se realiza ningún tipo de operación con estos datos y sólo son utilizados para evitar los efectos de la ISI y de la ICI, esta información será descartada.

Se realizará esta operación tantas veces como símbolos OFDM contenga la trama enviada que, como ya se ha dicho, consta de 100 símbolos.

En la simulación de la figura 3.13, se puede ver el paso del estado `Sincronización` a `QuitaPC`, observando que en este último sólo se modifica la dirección de memoria de la RAM del *Receptor* `dem_address`. Tras ello, se realiza la carga de datos al módulo *xFFT*.

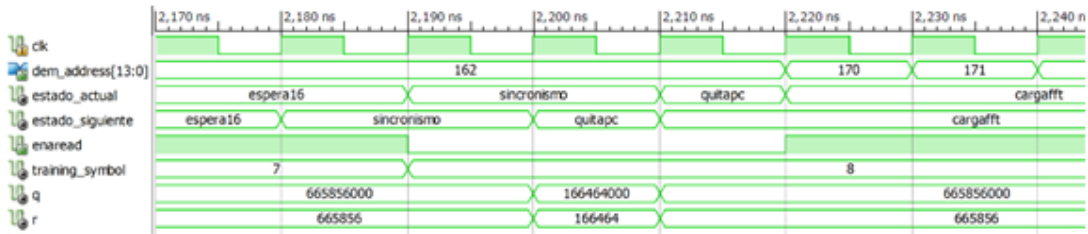


Figura 3.13: Simulación de la supresión del PC

Carga FFT

Como ya se hizo en el *Transmisor*, en este estado se cargarán los datos al módulo *xFFT*. Las operaciones realizadas son exactamente las mismas a las ya explicadas en bloque modulador 2.6 salvo por la señal `fwd_inv` que estará a nivel alto, indicando así el cálculo de una FFT directa (sección 2.4).

Cálculo FFT

Se permanecerá en este estado hasta que el cálculo de la FFT haya finalizado. Al igual que en *Transmisor*, esta operación finalizará cuando la señal `busy` esté a nivel bajo y la señal `done` cambie a nivel alto.

Por último, se activa la descarga de datos a través de la señal `unload`.

Su funcionamiento se puede comprobar en las simulaciones del *Modulador* o del bloque *xFFT* (secciones 2.6 y 2.4 respectivamente).

Almacena Simb

Tras el cálculo de la FFT, se guardarán los resultados en una memoria RAM. A diferencia del bloque *Modulador*, la memoria RAM empleada se ha generado a partir del *IP Core Block RAM Memory* como ya se hizo en la memoria del *Receptor*. Esta vez, dado que la lectura y escritura no se realiza simultáneamente, será de tipo *simple* y su tamaño estará limitado a 6400 posiciones, tantas como datos útiles de la trama OFDM.

Como ya ocurría en el *Modulador*, la descarga de datos no es inmediata y hay que esperar 8 ciclos hasta su inicio. Después de esta espera, se irán guardando en la memoria uno a uno cada resultado de la FFT.

En la simulación de la figura 3.14, se puede ver que, una vez que se han esperado los 8 ciclos (señal `contador`), se inicia la descarga de datos, aumentado en cada ciclo de reloj la dirección de memoria `direccionSimb`.

Si se trata del último símbolo de la trama se pasará a realizar la ecualización de la misma. En caso contrario, habrá que suprimir el prefijo cíclico del siguiente símbolo y volver a realizar el cálculo de la FFT hasta haber finalizado con los 100 símbolos de la trama enviada.

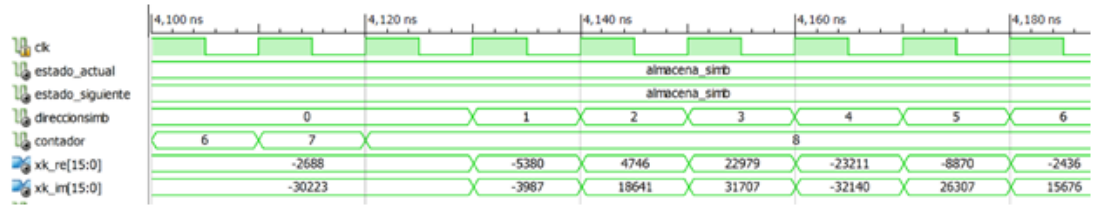


Figura 3.14: Simulación del almacenamiento de símbolos en la memoria *RAM_Simb*

Ecualización

En este estado se ejecutan todos los procedimientos explicados en el capítulo 5 para efectuar la ecualización de la trama de datos.

Una vez realizada la ecualización, se pasará al estado `obteniendo_bits`, finalizando así las operaciones del bloque *Receptor*.

Obteniendo Bits

Finalmente, una vez que la trama está ecualizada, se realizará la asociación símbolo-bit, concluyendo así el proceso de demodulación.

Una vez terminada la recuperación de bits, se pasará al estado `fin` antes de volver a iniciar los procesos de sincronismo y demodulación (estado `inicio`).

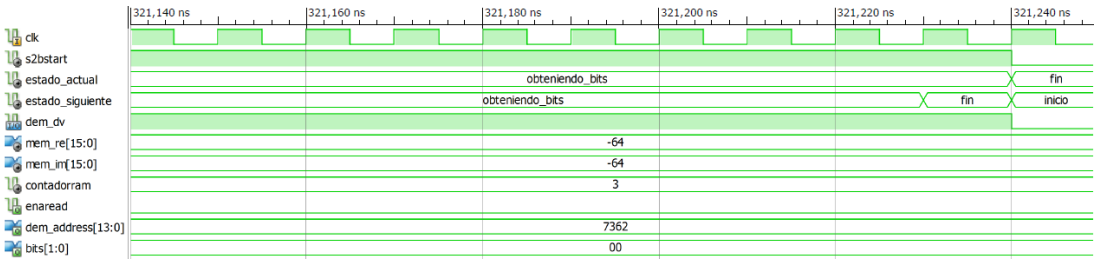


Figura 3.15: Simulación de la recuperación de bits

En la simulación de la figura 3.15 se puede ver desde la asociación símbolo-bit, hasta el paso por el estado `fin`.

Por último, conocidos todos los bloques que componen el sistema receptor se presenta en las imágenes de las figuras 3.16 y 3.17 el conexionado de bloques del mismo.

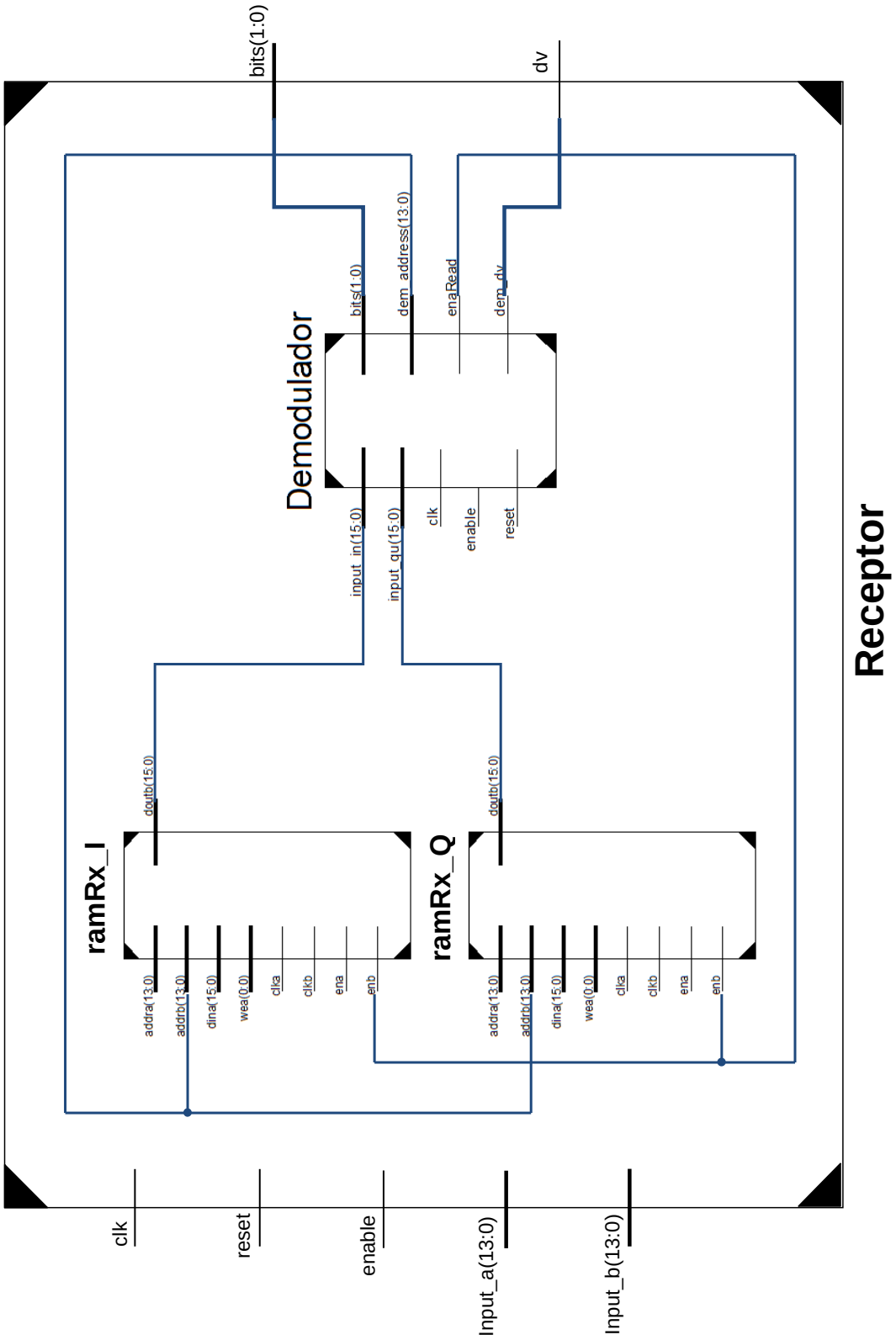


Figura 3.16: Diagrama de bloques del *Receptor*

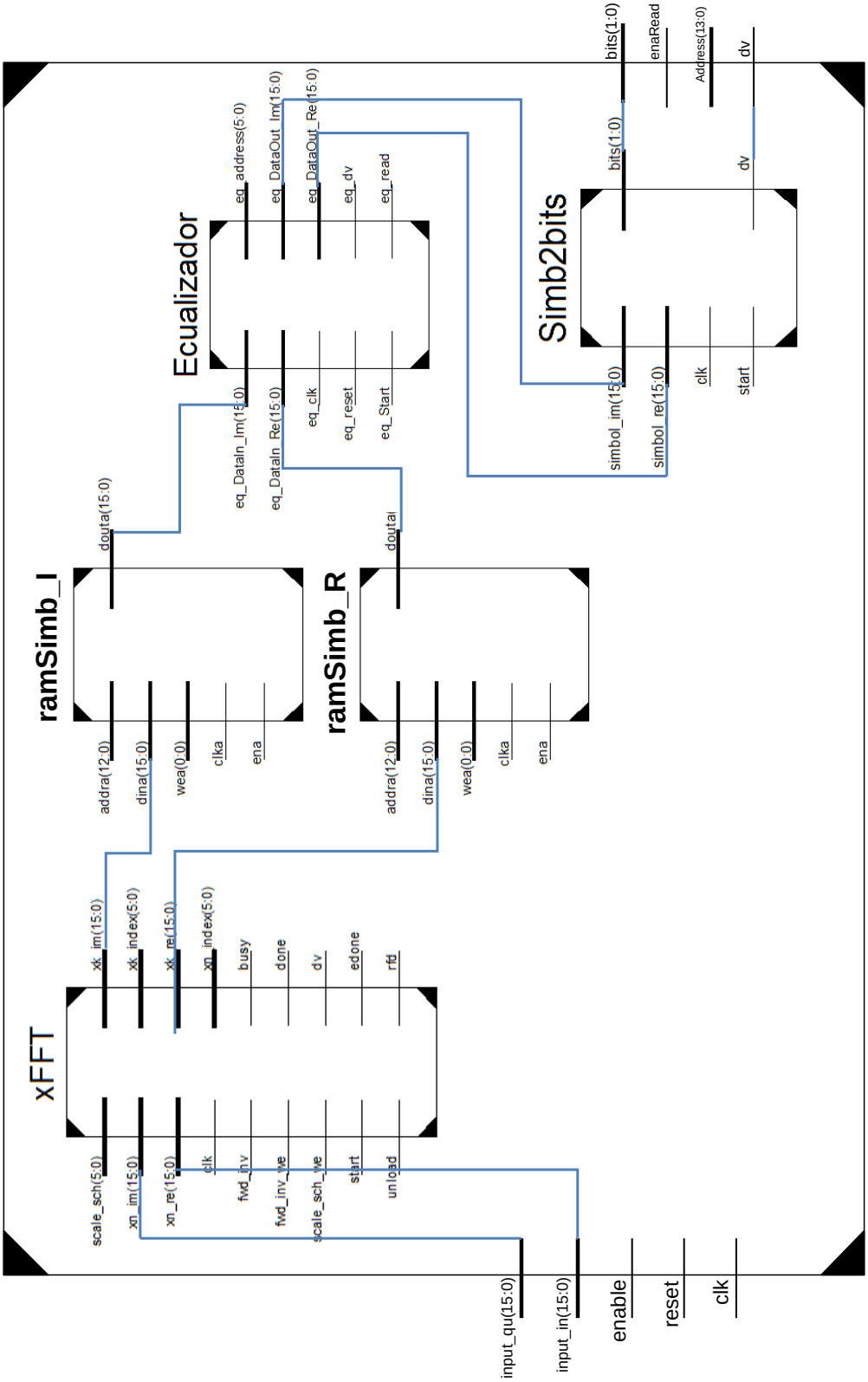


Figura 3.17: Diagrama de bloques del *Demodulador*

Capítulo 4

Sincronismo

En este capítulo se va a explicar todo el proceso de sincronización empleado en el sistema. En primer lugar, se comenzará con una breve descripción de las señales de referencia introducidas siguiendo el estándar 802.11a. A continuación se pasará al sistema implementado, donde se explicará la métrica utilizada para conseguir la sincronización temporal. Finalmente, se mostrarán unas simulaciones que demuestran el buen funcionamiento de este módulo del sistema.

4.1. Sincronización Temporal

Como ya se ha explicado anteriormente, la sincronización temporal es fundamental para conocer el inicio de la trama OFDM y poder así continuar con la demodulación de la misma. A partir de una métrica y de la secuencia corta de entrenamiento, se podrá llevar a cabo esta operación.

En la sección 1.5, se pudo ver que nuestra trama constaba de unos preámbulos PLCP, mostrados en la figura 1.7, cuya función entre otras era la de sincronización temporal. Estos preámbulos a su vez constaban de dos secuencias de entrenamiento y será precisamente la STS la que se utilizará para encontrar el inicio de la trama de datos.

Como se ha dicho, además de la STS se emplea una métrica. La métrica M no es más que la autocorrelación de cada uno de los símbolos que forman parte de la STS, con una longitud igual a L, normalizados por su energía como se puede comprobar en las ecuaciones 4.1, 4.2 y 4.3.

$$M(n) = \frac{|Q(n)|^2}{(R(n))^2} \tag{4.1}$$

$$Q(n) = \sum_{m=0}^{L-1} (r^*_{n+m} \cdot r_{n+m+L}) \tag{4.2}$$

$$R(n) = \sum_{m=0}^{L-1} |r_{n+m+L}|^2 \tag{4.3}$$

donde n es el índice del tiempo, r_n es la muestra en el instante n y L el número de muestras de la segunda parte del símbolo.

El algoritmo utilizado es el siguiente: en primer lugar se toman $2L$ muestras de nuestra supuesta trama y se realiza la métrica. Si el valor de dicha métrica es mayor que un umbral previamente calculado, se considerará que estamos ante un símbolo corto de entrenamiento. En caso contrario, se desplazará la ventana de $2L$ muestras una muestra y se realizará de nuevo la misma operación.

El algoritmo finalizará cuando se hayan detectado 10 símbolos cortos de entrenamiento seguidos, es decir, que la métrica calculada haya dado como resultado 10 valores mayores que el umbral de manera consecutiva.

Por último, sólo queda comentar los valores de la ventana L y del umbral. En [5], se considera 16 el valor óptimo de L, coincidiendo con el tamaño de los símbolos de la secuencia corta. Por otra parte, determinar el valor del umbral resulta fundamental, pues un umbral muy bajo daría lugar a una probabilidad de falsa alarma muy elevada y, por el contrario, un umbral alto haría que la probabilidad de pérdida de paquetes se disparase. Por tanto, se considera un valor entre $0{,}6-0{,}8$ óptimo en función de si estamos o no en un canal multitrayecto.

4.2. Implementación VHDL

Como ya se explicó en la sección 4.1, el algoritmo utilizado para la sincronización consistía en tomar 32 muestras y hacer la autocorrelación de las 16 primeras muestras con las otras 16 y normalizarlas con la energía de las últimas 16 muestras. Si este cociente da mayor que el umbral 10 veces consecutivas entonces se ha detectado el inicio de la trama.

Uno de los problemas que surge en este bloque del sistema es la división ya que se trata de una operación bastante compleja en hardware. Si el divisor podemos expresarlo como una potencia de 2, 2^n, el problema se resuelve rápidamente, pues la división no sería más que un desplazamiento de n bits a la izquierda.

Sin embargo, el valor de la energía del símbolo no siempre va a resultar potencia de 2 de ahí que se optase por usar un *IP Core* de *Xilinx* para realizar dicha operación. El principal inconveniente que conlleva utilizar este *Core* es el retardo que ocasiona. Como bien se explica en [7], la latencia generada es igual al número de bits del dividendo y, aunque en un primer momento se piense que dicho valor es igual a 16, en realidad no es así.

Al tratarse de una normalización, el resultado obtenido estará comprendido entre 0 y 1. El problema surge debido a que las operaciones se están realizando con números enteros por lo que los resultados generados sólo tienen dos valores posibles: 0 ó 1. Para evitar esta situación, se realiza una amplificación, multiplicando por 1000 el numerador, consiguiendo así resultados entre 0 y 1000.

Al amplificar por 1000 el dividendo, se necesitarían más de 40 bits para poder expresar el resultado, lo que daría lugar a un retardo de 40 ciclos por cada división, haciendo no sólo que el tiempo de procesamiento del receptor sea muy elevado sino también un aumento de los recursos del sistema al necesitar memorias de mayor tamaño con el fin de evitar que el *Receptor* sobrescriba los datos que aún están por procesar.

Finalmente, con el fin de evitar la división, se ha preferido usar la comparación de la ecuación 4.4.

$$Q(n) > Th \cdot R(n) \tag{4.4}$$

Si la autocorrelación de las L muestras, amplificada por 1000, es mayor que el producto del umbral (Th) por la energía de las mismas ($R(n)$), entonces se estará ante la posibilidad de un inicio de trama.

Con esta operación, por tanto, se consigue no sólo reducir la latencia que conlleva realizar una división, sino también el número de recursos de la FPGA al evitar una operación de gran coste computacional y posiciones de memoria extra en la memoria inicial del *Receptor*.

Por último, quedaría por fijar el umbral. En las simulaciones que se irán mostrando a lo largo de esta sección el umbral seleccionado ha sido de 900 aunque perfectamente se podía haber usado 1000 pues no se están teniendo en cuenta

los efectos del canal. En cuanto a su implementación práctica, como se verá en el capítulo 6, se han obtenido resultados satisfactorios con un umbral de 800.

Para realizar este algoritmo, al igual que se ha hecho en anteriores ocasiones, se ha utilizado una máquina de estados como la de la figura 4.1.

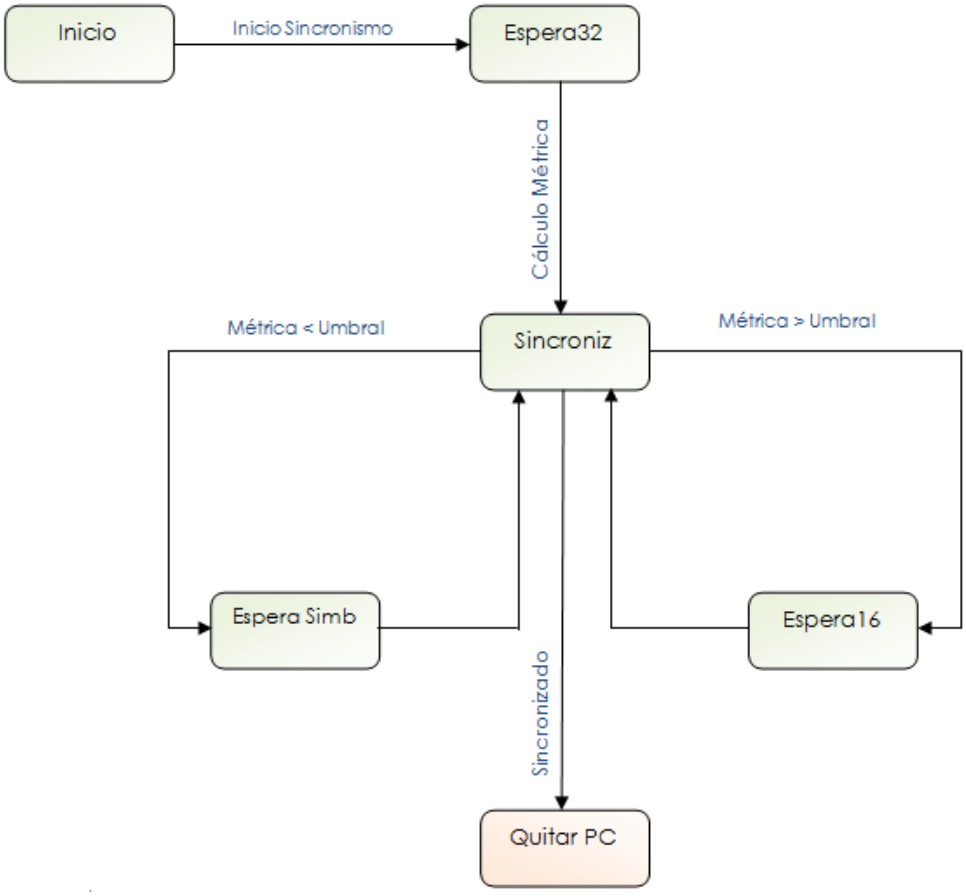


Figura 4.1: Diagrama de estados del bloque *Sincronismo*

Como se puede ver en la figura 4.1, todos los estados en color azul formarían parte del bloque sincronismo de la máquina de estados del *Demodulador* mostrada en la figura 3.12. Por tanto, una vez conseguido el sincronismo, se enlazará con la máquina de estados del *Demodulador*, comenzando por eliminar el PC del primer símbolo OFDM.

Inicio

En este estado se inicializan todas las variables y se permanecerá a la espera de que se inicie el proceso de sincronización de la trama.

Espera32

La función de este estado es la de tomar las 32 muestras de las memorias RAM antes de iniciar el cálculo de la métrica.

La figura 4.2 muestra como una vez que se tienen las 32 muestras (señal `contador`) almacenadas en la matriz `sinc`, se pasa al estado `sincronismo`.

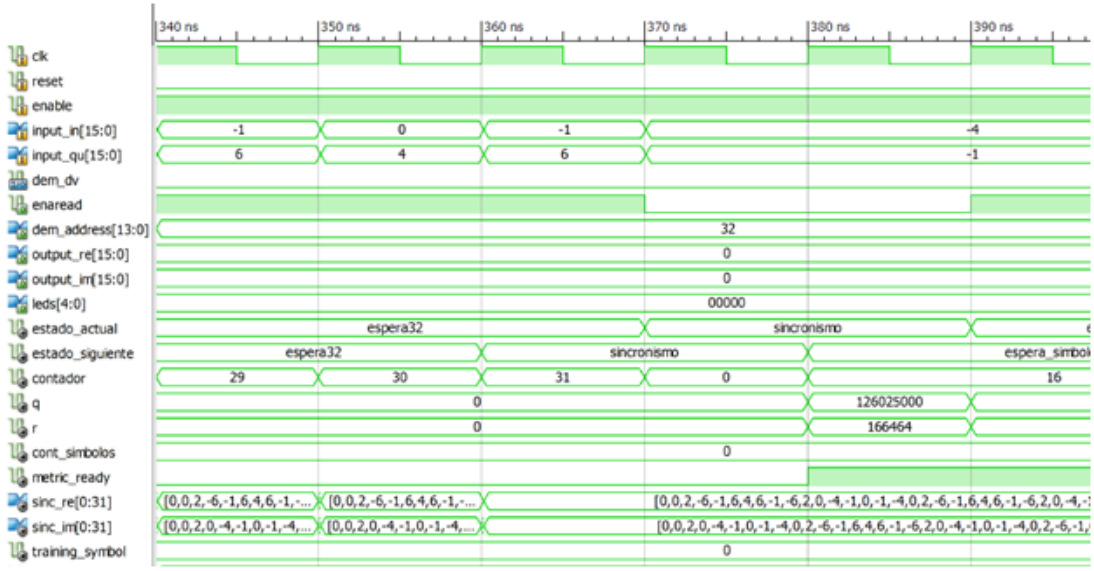


Figura 4.2: Simulación del bloque *Sincronismo*

Sincronización

En este estado se realiza el cálculo de la métrica.

Como ya se explicó, si este valor es inferior al producto del umbral por la autocorrelación de las 16 últimas muestras, habrá que desplazar la ventana de 32 muestras una posición. En la simulación de la figura 4.2, se puede comprobar

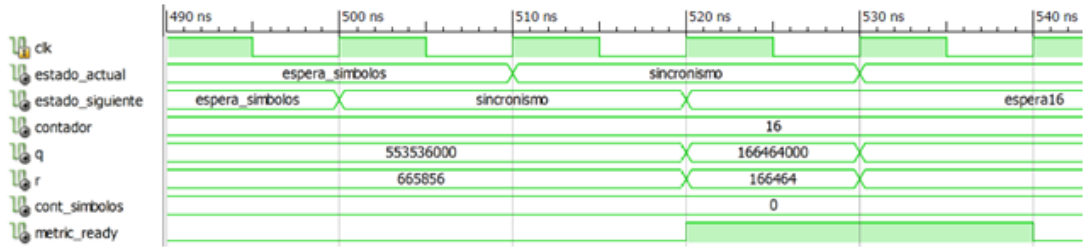


Figura 4.3: Simulación del cálculo de una métrica mayor que el umbral

cómo, para un umbral fijado en 900, la métrica es inferior al umbral por la autocorrelación, por tanto, se pasará al estado `espera_simbolos` para realizar el desplazamiento de la ventana.

Por el contrario, si el resultado es mayor, se habrán encontrado dos STS, por lo que se desplazará la ventana 16 muestras para comprobar si el siguiente símbolo es también un STS. En la figura 4.3, se observa cómo la métrica en este caso es mayor que el producto entre la autocorrelación y el umbral, pasando al estado `espera16`.

Por último, si se han encontrado los 10 STS se habrá conseguido la sincronización pudiendo así realizar la demodulación de la trama. En la simulación de la figura 4.4, se observa cómo la señal `cont_simbolos` marca 8, pasando así a la demodulación.

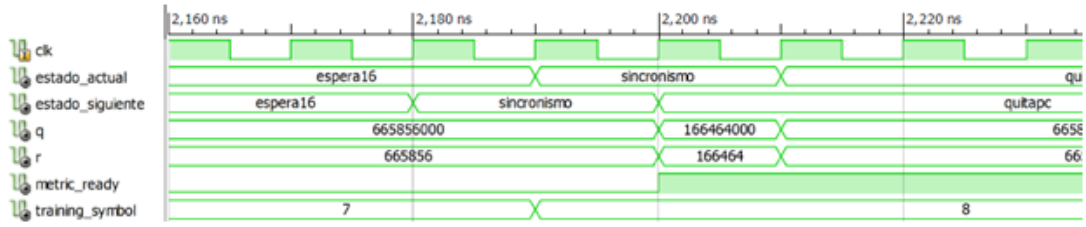


Figura 4.4: Simulación del cálculo de la métrica entre los dos últimos símbolos STS

EsperaSimb

Se pasará por este estado siempre que la métrica no supere el umbral. Cuando esto ocurra, se desplazará una muestra la ventana de 32 y se volverá a calcular la métrica. Es decir, se desplazarán todos los valores de la matriz `Sinc` y se añadirá al final uno nuevo procedente de la memoria RAM del *Receptor*.

En la figura 4.2, se puede ver cómo la métrica es inferior, pasando así a `EsperaSimb`.

Espera16

Si la métrica es mayor que el umbral se pasará por este estado, en el cual, la ventana se desplaza 16 muestras, haciendo que la segunda parte del símbolo se convierta en la primera, y se vuelve a calcular la métrica. Al igual que en `EsperaSimb`, se desplazarán todos los valores de la matriz `Sinc`, sólo que esta vez el desplazamiento es de 16 muestras, por lo que habrá que leer 16 valores de la memoria RAM del `Receptor`.

En la figura 4.3, se puede ver el paso de `sincronismo` a `Espera16`.

Una vez que se ha realizado la sincronización ya se puede operar con la trama, iniciando la supresión del prefijo cíclico.

Como se ha podido comprobar, la sincronización es un proceso bastante costoso que requiere la manipulación de cada una de las muestras almacenadas en la memoria RAM del *Receptor*. La latencia provocada por esta operación es difícil de estimar, pues un error en algún símbolo del preámbulo puede originar la pérdida total del paquete.

Capítulo 5

Ecualización

En este capítulo se estudiará el ecualizador. Su diseño está basado en el TFG [8], en el cual ya se realiza una comprobación en Matlab. Primero se comenzará con su descripción teórica para terminar con su implementación en VHDL contando todas las limitaciones que ello conlleva. Además se mostrarán las simulaciones realizadas con el fin de comprobar el funcionamiento del mismo.

5.1. Descripción

Como ya se ha explicado en la sección 1.6, la ecualización trata de corregir esas variaciones que aparecen en nuestros símbolos debido a su paso por el canal. Para ello, se utilizan una serie de técnicas de estimación de canal basadas en métodos de interpolación.

A continuación, se pasará a explicar el algoritmo seguido en el proceso de ecualización de la trama de datos. Nuestro propósito es el de conseguir una matriz H con todos los coeficientes que multiplican a la señal transmitida, permitiéndonos así corregir las distorsiones provocadas por el canal.

El estimador implementado, como bien se ha dicho en la sección 1.6, es el LS o de *mínimos cuadrados*, de ahí la necesidad de buscar esa matriz H con la estimación de canal. Por tanto, a partir de esa matriz con las estimaciones y la matriz recibida, R_x, se podrá recuperar la información inicial.

$$Rx = Tx \cdot H + N \implies \hat{T}x = \frac{Rx - N}{H} \tag{5.1}$$

Dicha recuperación de la información se realiza como se muestra en la ecuación 5.1, es decir, no es más que la división de la matriz de datos recibida, R_x, entre la matriz H con la estimación de canal.

En cuanto al ruido, dado que también se verá afectado por la matriz H, la SNR del sistema permanecerá constante. Por otra parte, como ya se explicó, el estimador elegido no es el más robusto en cuanto a ruido, pero debido a la sencillez del LS y las limitaciones de nuestro dispositivo, se optó por asumir esta degradación.

El algoritmo seguido [8] se puede dividir en dos fases. En la primera, se realiza una estimación en el dominio de la frecuencia utilizando una interpolación basada en el uso de la IFFT/FFT. Tras ella, se pasa a la estimación en el dominio del tiempo, utilizando en este caso una interpolación lineal.

La estimación en el domino de la frecuencia consta a su vez de cuatro pasos previamente explicados. En primer lugar, habrá que generar un vector P con las señales pilotos de un determinado símbolo OFDM. En la figura 5.1 se puede ver un ejemplo de la generación del vector P a partir de las señales pilotos del primer símbolo.

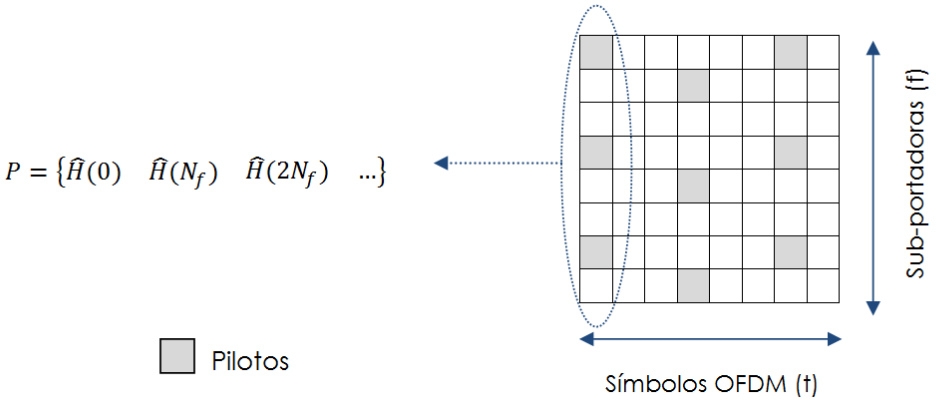


Figura 5.1: Ejemplo de matriz P

A continuación se pasa al dominio temporal el vector P realizando la IDFT, obteniendo así el vector $\hat{h}_s$.

$$\hat{h}_s = IDFT(P) \tag{5.2}$$

Una vez obtenido el vector $\hat{h}_s$, hay que calcular el vector $\hat{h}$ que no es más que el relleno de ceros del vector $\hat{h}_s$, de tal forma que su longitud total sea igual al número de portadoras N.

$$\hat{h} = \{\hat{h}_s \; 0 \; 0 \; ... \; 0 \; \} \tag{5.3}$$

Por último, habrá que volver a pasar al dominio frecuencial, obteniendo así el vector $\hat{H}$ mediante la DFT de $\hat{h}$.

$$\hat{H} = DFT(\hat{h}) \tag{5.4}$$

Este vector $\hat{H}$ se corresponde con una columna de la matriz H, por lo que se tendrá que volver a repetir este algoritmo tantas veces como columnas con señales de referencia se hayan insertado en el transmisor.

Realizada la interpolación en el dominio de la frecuencia, se puede pasar a la interpolación en el dominio temporal. Tal y como se ha explicado, la técnica de interpolación escogida en este caso es la interpolación lineal.

A partir de las columnas calculadas previamente, se comienza a calcular el resto de valores de la matriz H. Recorriendo cada columna, se irá aplicando la ecuación 1.7, asignando los valores Y_1, Y_2, X_1 y X_2 en función de su posición. En la figura 5.2 se puede ver la representación de cada uno de estos valores.

El mayor inconveniente de este tipo de interpolación es la necesidad de conocer el valor de los extremos del intervalo en el que se quiere realizar esta operación. Esto limitará los posibles valores para la separación temporal entre los pilotos, N_t, que, como ya se explicó en la sección 2.3 debe ser 3, 9, 11 y 33 para que con una trama de 100 símbolos OFDM se tengan señales de referencia en el primer y último símbolo.

Completada la matriz H, ya se puede corregir la trama recibida R_x dividiéndola por la matriz H.

A continuación, se explicará la implementación en VHDL de todo este proceso, mostrando, como ya se ha ido haciendo a lo largo de este trabajo, una serie de simulaciones confirmando el correcto funcionamiento.

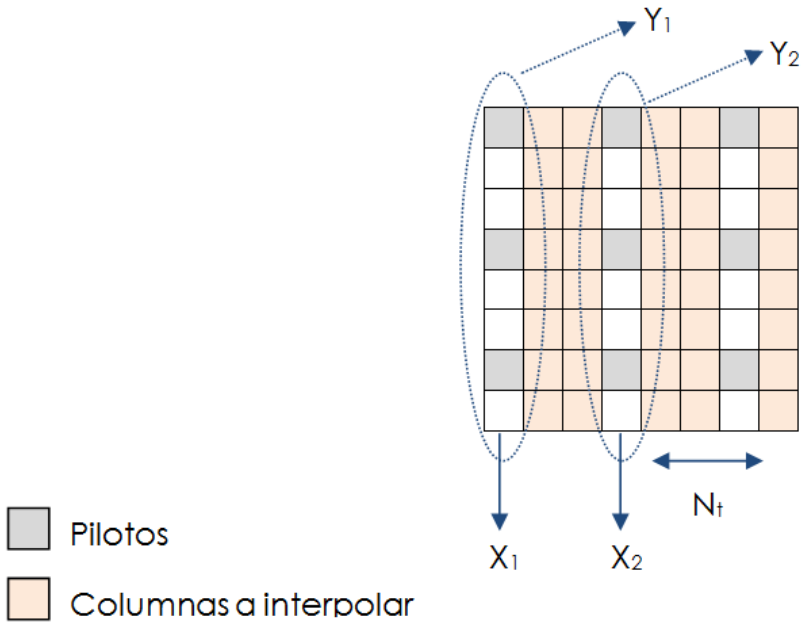


Figura 5.2: Representación de los valores de la interpolación lineal

5.2. Implementación en VHDL

Con lo anteriormente descrito ya se puede diseñar el bloque *Ecualizador*. Será un bloque síncrono cuyas entradas serán los datos procedentes de la memoria *RAM_Simb* del *Demodulador* por lo que como salida tendrá de que tener la dirección de memoria que se pretende leer además de la señal de habilitación de lectura. Por otra parte, tendrá también como salida los símbolos ecualizados junto con la señal que indica que hay datos válidos a la salida.

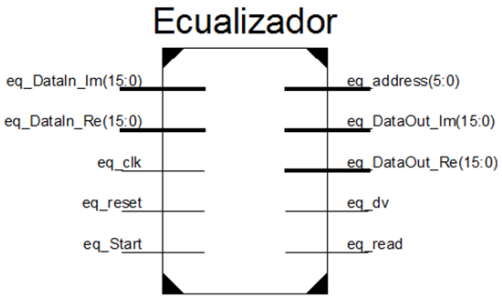


Figura 5.3: Diagrama de bloques de *Ecualizador*

En la figura 5.3 podemos ver el diagrama de bloques del ecualizador.

Interfaces de entrada:

- `clk`: reloj del sistema
- `reset`: señal asíncrona
- `eq_start`: señal de inicio
- `eq_DataIn_Re/eq_DataIn_Im`: Datos de entrada

Interfaces de salida:

- `eq_address`: posición de memoria deseada de la *RAM_EQ*
- `eq_read`: señal de activación del modo lectura en la *RAM_EQ*
- `eq_dv`: Data Valid. Indica que hay datos en la interfaz de salida
- `eq_DataOut_Re/eq_DataOut_Im`: Datos de salida

Como se ha ido viendo a lo largo de este trabajo, la implementación en VHDL del sistema anteriormente descrito está basada en una máquina de estados. Sin embargo, su implementación sufrirá una serie de modificaciones debido a las limitaciones del dispositivo.

En primer lugar, el algoritmo para la estimación de canal en el dominio de la frecuencia se mantendrá igual, es decir, primero se calculará el vector P para posteriormente convertirlo al dominio temporal con el fin de añadir tantos ceros como número de sub-portadoras N se hayan generado. Por último, se volverá a pasar ese vector al dominio frecuencial.

Sin embargo, la limitación que aparece en este bloque como ya se explicó en la sección 2.3 es la separación entre pilotos en el dominio de la frecuencia N_f. El *IP Core* de la FFT sólo permite realizar dicha operación para valores 2^m con $m = 3 - 16$. Por tanto, para un total de $N = 64$ portadoras sólo se podrán considerar una separación de 8, 16, 32 ó 64.

Por otra parte, para la estimación de canal en el dominio temporal ya se ha contado cuál es el factor limitante, la necesidad de que el primer y último símbolo de la trama contengan señales piloto.

El algoritmo implementado para el cálculo de la interpolación lineal es bastante simple. En primer lugar, se tomarán dos de las columnas que contienen la estimación de canal en el dominio de la frecuencia. A partir de estas dos columnas ya se pueden establecer los valores X_1, X_2, Y_1 e Y_2. Por último, se irán recorriendo todas las posiciones de la matriz contenidas entre esas dos columnas, calculando así la estimación en el dominio temporal usando la ecuación 5.1.

Sin embargo, como ya ocurrió con el sincronismo, la división no es una operación sintetizable lo que hace que se tenga que usar un *IP Core* provocando que el tiempo de demodulación de la trama aumente significativamente.

En la figura 5.4 se puede ver la máquina de estados implementada.

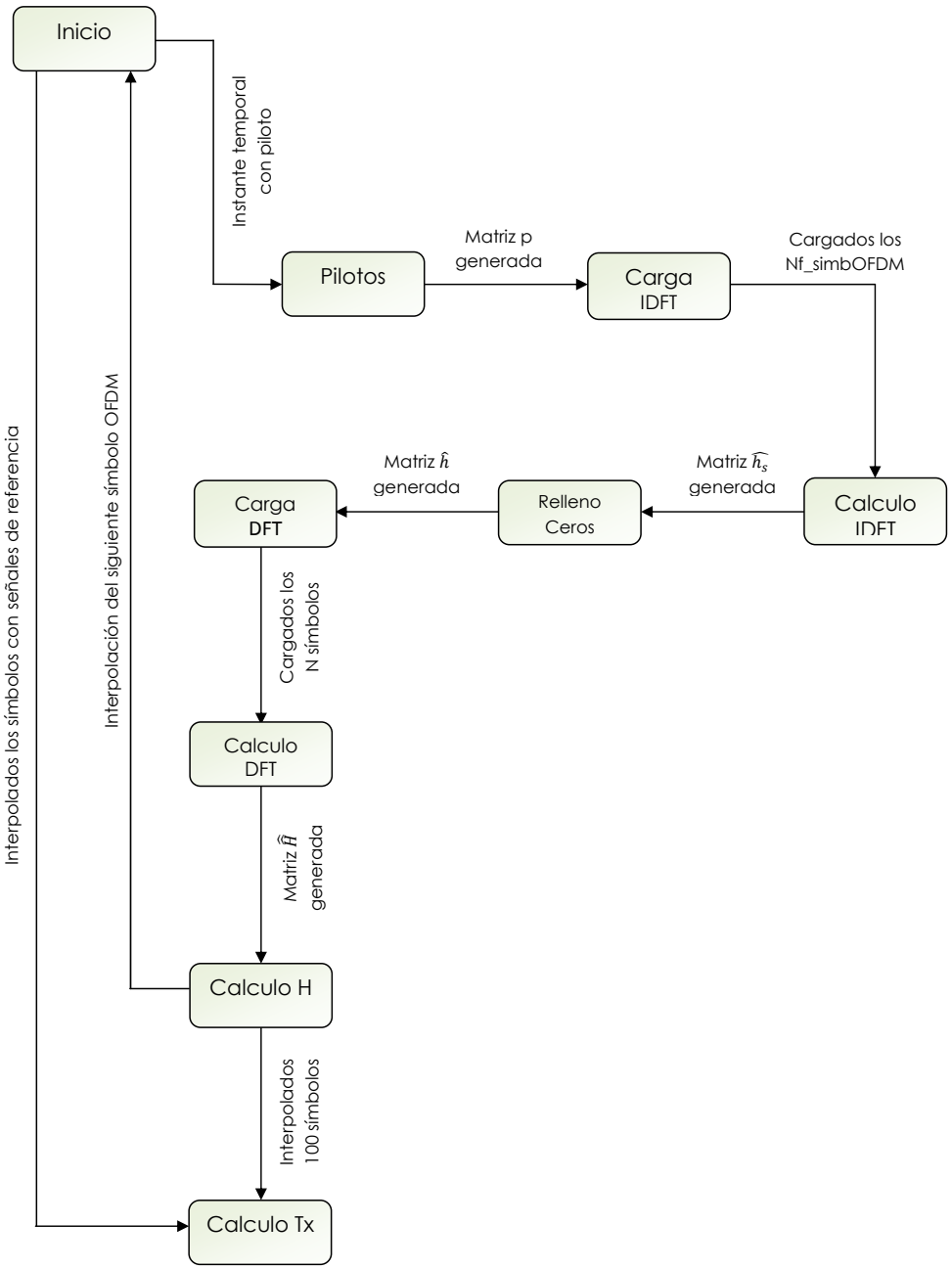


Figura 5.4: Diagrama de bloques del *Ecualizador*

Inicio

En este estado se realiza la inicialización de variables antes de comenzar con todo el proceso de ecualización. Dicho proceso se inicia cuando la señal `start` se active.

Pilotos

En este estado se calculan los vectores P con las señales de referencia. Para ello hay que acceder a la memoria *RAM_Simb* del *Demodulador* y leer exclusivamente esas direcciones de memoria. Una vez que se han rellenado los vectores, se realizará la estimación en el dominio de la frecuencia mediante la DFT.

En la simulación de la figura 5.5 se puede ver cómo se van rellenando los vectores P con la información obtenida de las memorias RAM. También se puede comprobar cómo la dirección de memoria sólo marca las posiciones con piloto, es decir, múltiplos de N_f. Como en este caso, el valor seleccionado para N_f es 8, `ram_address` toma valores 0, 8, 16, 24, etc.

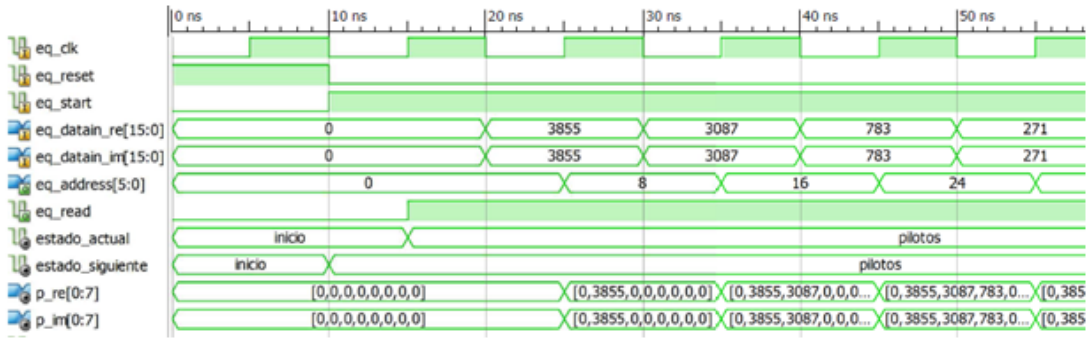


Figura 5.5: Simulación del estado `Pilotos`

Carga_IDFT

Como ya se ha hecho anteriormente, antes de realizar el cálculo de la IFFT hay que cargar cada uno de los datos al módulo de forma serie. El funcionamiento de este módulo ya se explicó en el apartado 2.4, la única diferencia reside en el tamaño de la FFT que será de N_f.

En la figura 5.6 se puede ver cómo se activan las señales de habilitación `ifft_start`, `ifft_fwd_inv_we` e `ifft_scale`, además de la carga de datos al módulo.

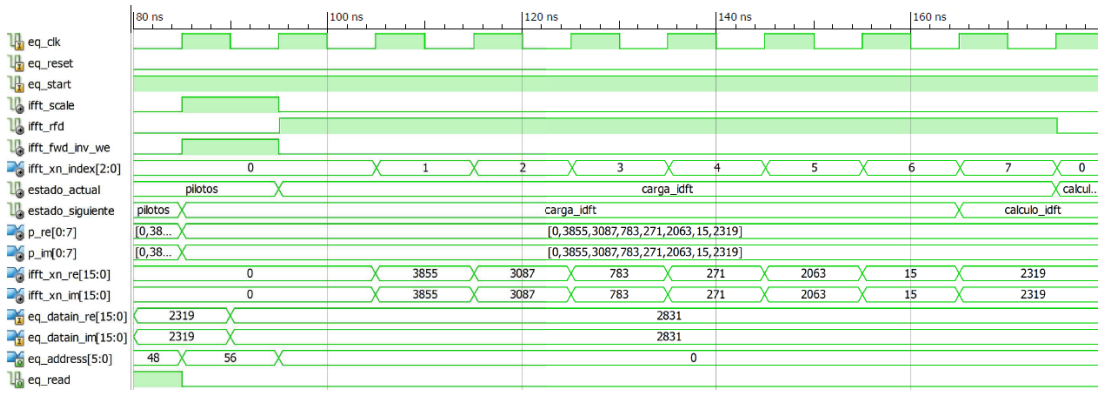


Figura 5.6: Simulación del estado `Carga_IDFT`

Calculo_IDFT

Se permanecerá en este estado hasta que las señales de fin de cálculo de IFFT se activen.

Relleno_Ceros

Una vez terminado el cálculo de la IFFT se procederá a la descarga de datos, guardando en el vector $\hat{h}$ el resultado de la IDFT del vector P. Como ya se ha explicado anteriormente, habrá que añadir tantos ceros al final del vector $\hat{h}$ como portadoras tenga nuestro sistema.

Para ello, se usaran los vectores `ceros_re` y `ceros_im`, los cuales son de tamaño igual al número de portadoras. En primer lugar, estos vectores estarán inicializados a 0, por lo que sólo habrá que ir guardando el resultado de la IDFT en las primeras posiciones del mismo.

En la simulación de la figura 5.7 se puede ver cómo el resultado de la IDFT es guardado en estos vectores, quedando las posiciones finales de los mismos con un valor igual a 0.

Carga_DFT

Siguiendo con el algoritmo, tras el relleno de ceros, se realiza la DFT del vector $\hat{h}$. Como ya se ha hecho en `Carga_IDFT` se habilitarán las señales de activación y se irán cargando uno a uno los valores del vector `Ceros`.

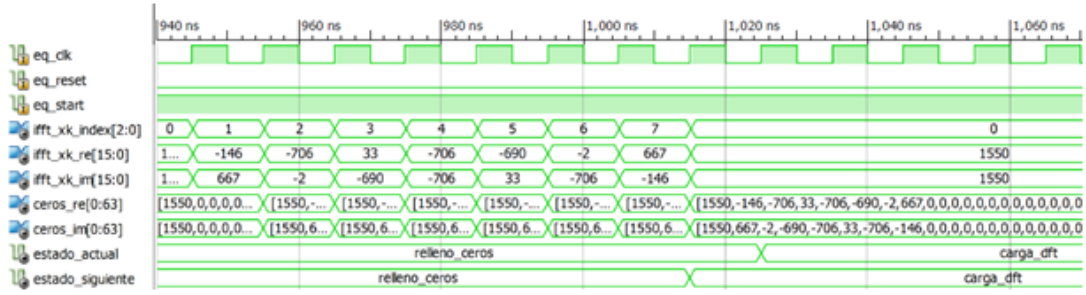


Figura 5.7: Simulación del estado `Relleno_Ceros`

Calculo_DFT

Se permanecerá en este estado hasta que las señales de fin de cálculo de FFT se activen.

Calculo_H

El resultado de la DFT del vector relleno con los ceros será guardado en la matriz H, ocupando así una columna de la misma.

En la figura 5.8 se observa cómo se van guardando los resultados de la DFT en la matriz H.

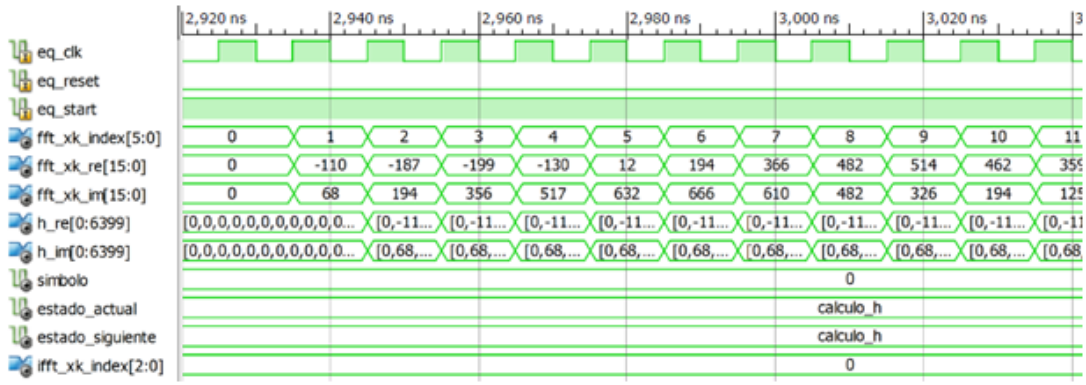


Figura 5.8: Simulación del estado `Cálculo_H`

Una vez que se tienen guardados los N símbolos, hay que comprobar si se está ante el último símbolo, si es así se iniciará la estimación en el dominio temporal. En

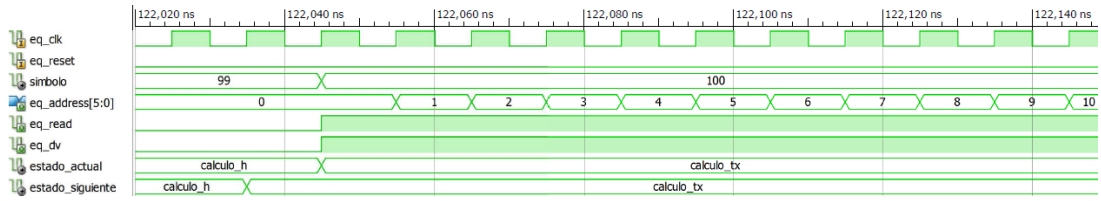


Figura 5.9: Simulación del paso del estado `Cálculo_H` a `Cálculo_Tx`

caso contrario, se volverá al estado `inicio` y se calculará la siguiente columna de la matriz de ecualización.

En la simulación de la figura 5.9 se puede ver cómo al llegar a los 99 símbolos (señal `simbolo`) se inicia el proceso de interpolación lineal. Debido a la dificultad de mostrar los resultados en la simulación, la figura 5.10 recoge algunos de los valores de la interpolación entre la primera y la cuarta columna de la matriz H, comprobando además de forma numérica los resultados obtenidos.

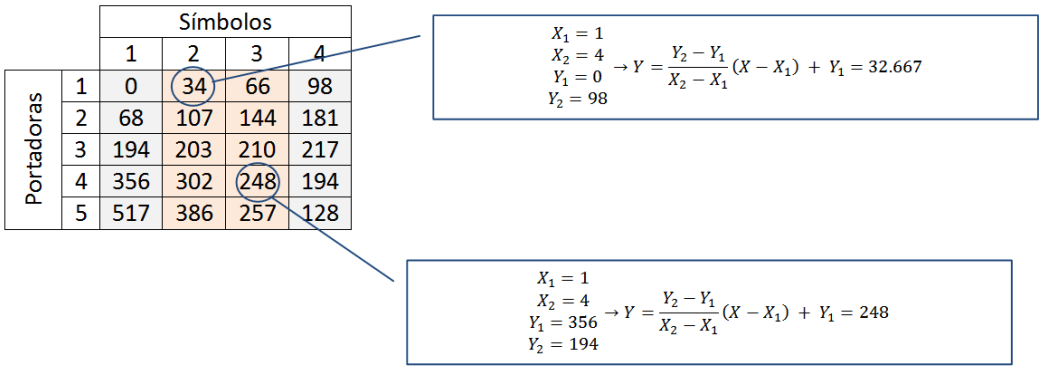


Figura 5.10: Resultados de la interpolación lineal

Calculo_Tx

Por último, queda el cálculo de la matriz transmitida. Siguiendo la ecuación 5.1 y al tratarse de un sistema SISO hay que dividir cada elemento recibido por su correspondiente en la matriz H. Para ello, se irán leyendo todos los elementos de la memoria *RAM_Simb* del *Demodulador* y dividiéndolos entre sus correspodientes en la matriz H.

En la simulación de la figura 5.9, se puede ver la lectura de datos de la memoria *RAM_Simb*. También se muestra la activación de la salida `eq_dv` con el fin de informar al bloque *Simb2Bits* que los símbolos están listos para convertirlos de nuevo a bits.

Por último, una vez que se han ecualizado todos los símbolos de la trama, se vuelve al estado inicial a la espera de la señal de activación `start`. La figura 5.11 muestra la ecualización de los últimos símbolos de la trama.

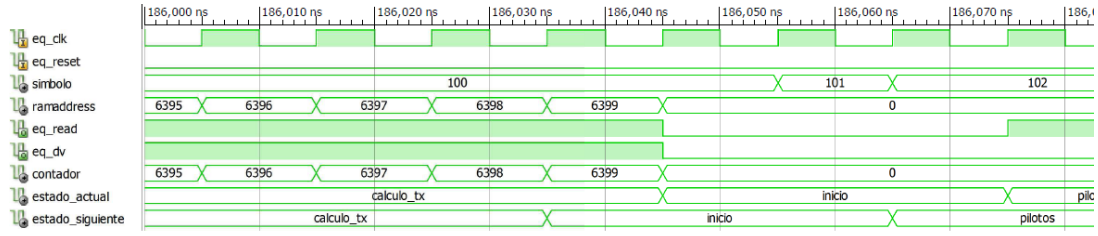


Figura 5.11: Resultados de la interpolación lineal

Capítulo 6

Pruebas y resultados

6.1. Entorno de trabajo

Para poder llevar a cabo la implementación del sistema de comunicaciones se han utilizado diferentes herramientas software y hardware, además de diferentes lenguajes de programación.

El dispositivo utilizado es la plataforma de desarrollo SFF SDR de la compañía *Lyrtech*. Se trata de un sistema concebido para el diseño y desarrollo de aplicaciones radio por software. Consta de tres módulos como podemos ver en la figura 6.1:

- Módulo de procesamiento digital
- Módulo de conversión de datos
- Módulo de radiofrecuencia

Sin embargo, a nivel de usuario se puede decir que está formado por dos módulos principales: un procesador digital de señales o DSP y una FPGA.

El DSP es el encargado de configurar los tres módulos anteriormente comentados. Establece la frecuencia de transmisión, la tasa de conversión de datos, el ancho de banda del filtro de IF entre otros parámetros.

La configuración se hace mediante un fichero en lenguaje *C* y la herramienta software *Code Composer Studio 3.3*, ya que gracias a ella el fichero fuente *C*

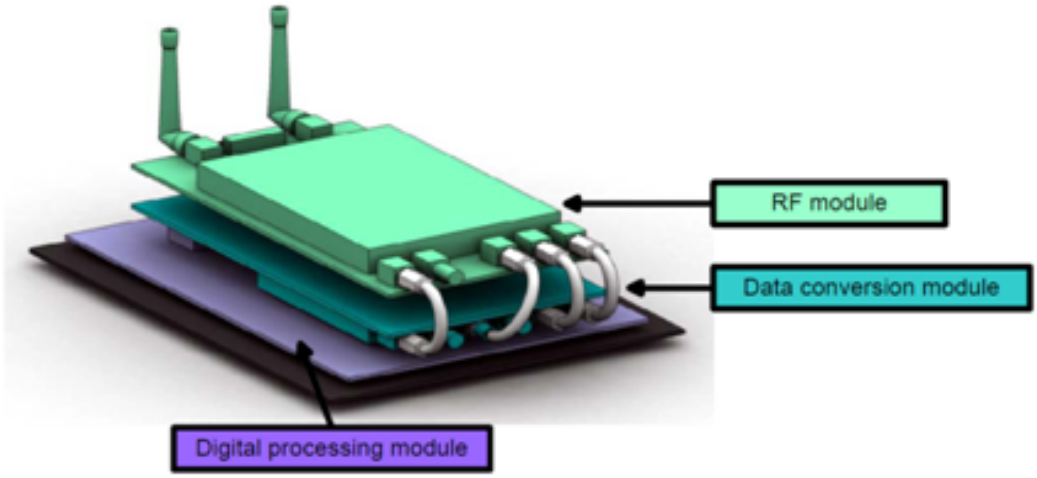


Figura 6.1: Módulos del dispositivo SFF SDR

será compilado, ensamblado y enlazado, generando así el fichero ejecutable `.out` utilizado por el dispositivo.

Por su parte, la FPGA empleada es una *Virtex-4 XC4SX35* y será en ella en donde se cargará nuestro diseño. El lenguaje empleado es VHDL con el fin de poder realizar una descripción del circuito electrónico que deseamos implementar.

Para llevar a cabo el desarrollo del código VHDL se utiliza la herramienta de desarrollo *Xilinx ISE 14.7*. Esta herramienta permite tanto el diseño, síntesis e implementación, generando el fichero `.bit` que será cargado en el dispositivo. Este programa además de las funciones anteriormente citadas permite la simulación gracias a su herramienta *iSim*, permitiendo de esta forma el análisis y la comprobación del diseño realizado antes de ser cargado en la placa.

Una vez finalizado todo el proceso de diseño e implementación, se cargan en la placa los ficheros ejecutables `.out` y `.bit` y empleando además el osciloscopio *Agilent Infiniium DSO90604A* se podrán medir las señales transmitidas y recibidas en el dispositivo.

El objetivo de esta sección era dar una visión básica de los dispositivos y software empleados, por ello si se desea tener más información sobre los mismos acuda a [9],[10],[11],[12] [13] o [14] entre otros.

6.2. Pruebas en el Transmisor

En esta sección se van a mostrar los resultados obtenidos en el bloque *Transmisor*. Cada una de las tramas enviadas consta de 100 símbolos OFDM con un total de 64 portadoras, con un prefijo cíclico de longitud 8 y una cabecera PLCP formada por una secuencia corta de entrenamiento con 10 símbolos.

Por otra parte, el reloj del sistema está en los 4,687 MHz, con una frecuencia de trabajo del DAC de 125 MHz.

Con estos datos ya se puede predecir la duración de una trama OFDM. Si se envían 100 símbolos con un total de 64 portadoras, hacen un total de 6400 símbolos, a los que hay que sumar los 160 símbolos de la STS y los 100 prefijos cíclicos de longitud 8. Esto hace un total de 7.360 símbolos por trama que, teniendo en cuenta la frecuencia de trabajo, hace que la duración de la misma sea de $1,57ms$:

$$\frac{(100 \times 64 + 160 + 100 \times 8)}{4,687MHz} = 1,5703ms \tag{6.1}$$

En la figura se muestra una trama de 100 símbolos en banda base. Como se puede ver la duración de la misma es de 1,577 ms, acercándose bastante al valor teórico previamente calculado.

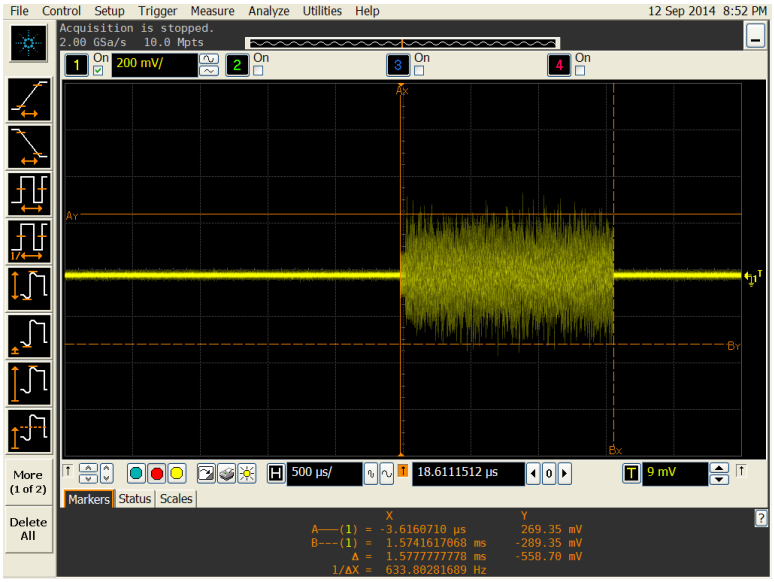


Figura 6.2: Trama OFDM en Banda Base

En la figura 6.2 también se puede observar que al inicio de la trama hay una serie de símbolos con una menor amplitud. Se trata de la secuencia corta de entrenamiento. En la figura se ha ampliado esa parte y se puede comprobar que, efectivamente, está formada por la repetición de 10 símbolos.

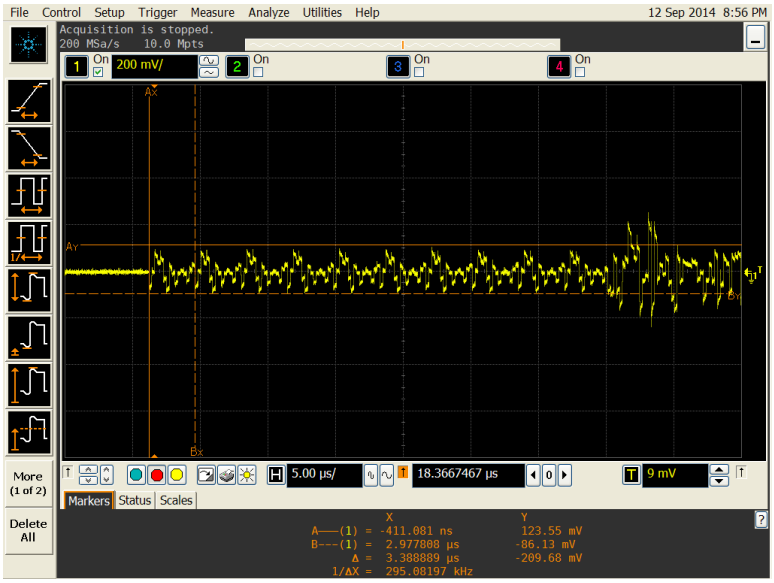


Figura 6.3: Secuencia corta de entrenamiento (STS)

De igual manera, si se calcula el tiempo de transmisión de un símbolo STS, este será igual 3,41 μs, coincidiendo con el valor obtenido en el osciloscopio.

Como se ha comentado, la amplitud de estos símbolos es menor, pasando de los 558,7 mV de los símbolos OFDM a los 289,6 mV.

Sin embargo, el mayor problema de la trama generada es el tiempo de procesamiento. En la figura 6.4 se observa que este tiempo es de 8,05 ms, incluso mayor que el tiempo de transmisión. No obstante, este valor coincide con el esperado, pues la simulación daba un tiempo de procesamiento bastante elevado.

A continuación, en la figura 6.5 se muestra el espectro de la señal en banda base, en la que se puede constatar que el ancho de banda es 4,687 MHz, con una diferencia entre lóbulos de 11,6 dB.

Por último, queda por mostrar la señal transmitida en RF. Tanto la amplitud como la duración de la señal no variarán con respecto a lo visto en banda base. En cuanto al espectro, mostrado en la figura 6.7, se puede ver que se sitúa a una frecuencia de 260 MHz, con un ancho de banda de 9,375 MHz, el doble de banda base.

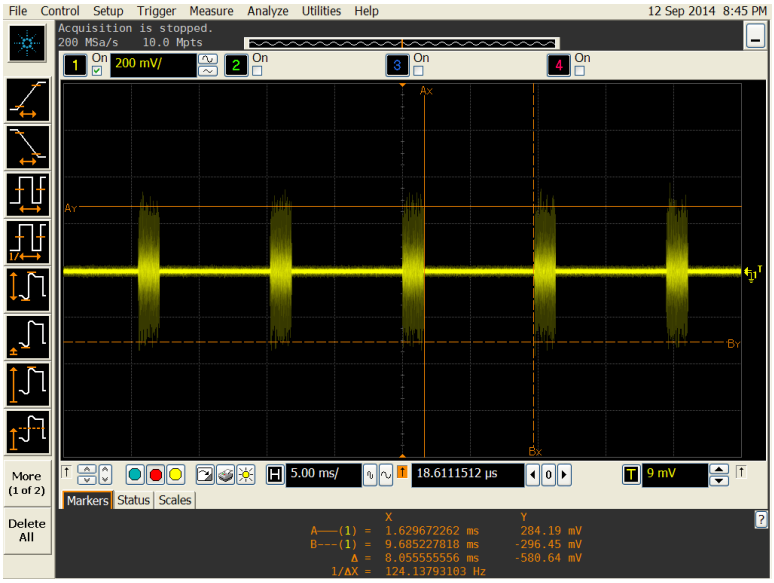


Figura 6.4: Tiempo de procesamiento del sistema

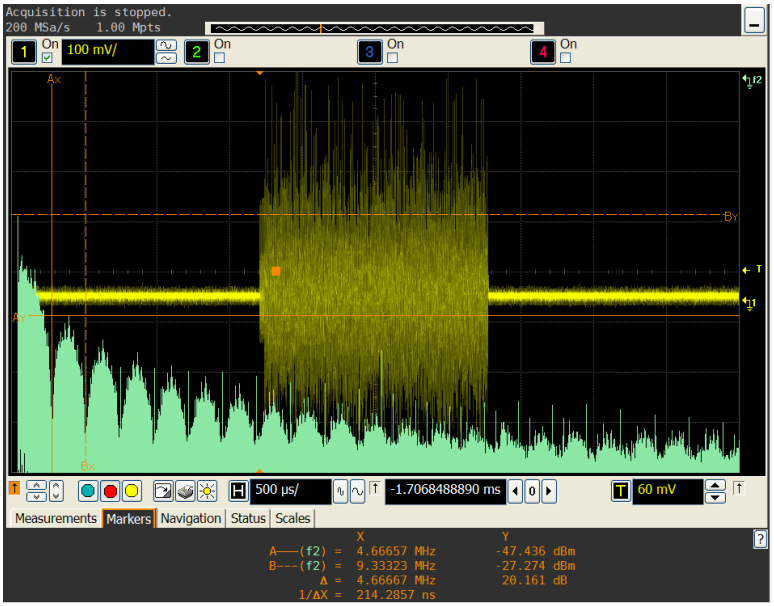


Figura 6.5: Espectro de la señal en banda base

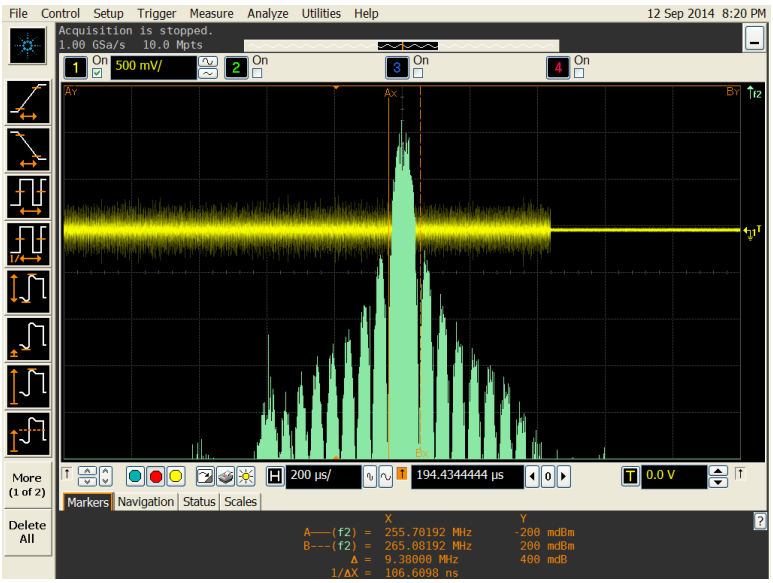


Figura 6.6: Espectro de la señal en RF

6.3. Pruebas en Recepción

A continuación se expone la señal recibida con el fin de mostrar las diferencias con respecto a la transmitida.

Como se puede ver en la figura 6.7, aparece el espectro de la señal en torno a los 30 MHz que es a la frecuencia a la que el módulo de radiofrecuencia convierte la señal recibida.

6.4. Pruebas de Sincronización

El mayor inconveniente de este dispositivo es la ausencia de puertos de salida para la señal recibida. Tan sólo se tiene acceso a la señal en IF, lo que dificulta el control de los datos demodulados. La única posibilidad es utilizar 5 LEDs del dispositivo y transmitir una secuencia de datos simple para ser capaces de reconocerla a través de estos LEDs.

Sin embargo, para las pruebas de sincronización, dado que los LEDs no proporcionan un mecanismo fiable para conocer si se ha sincronizado o no la señal, se ha optado por realizar el siguiente algoritmo.

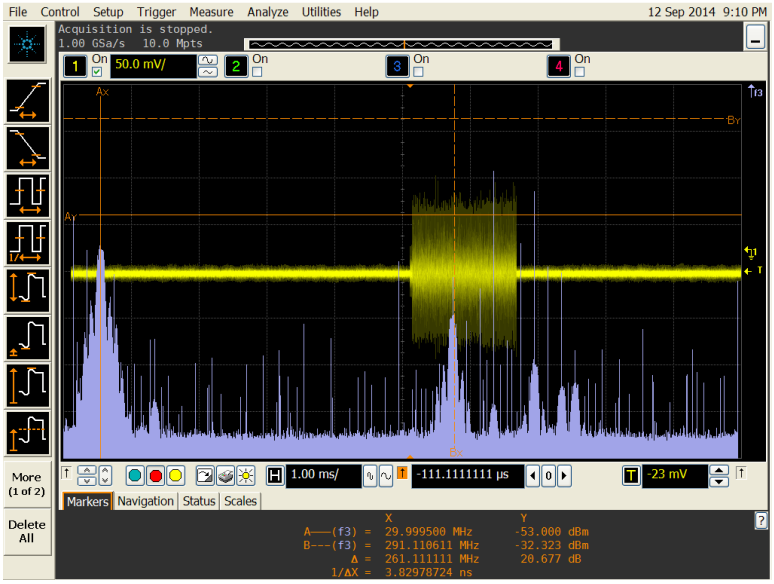


Figura 6.7: Espectro de la señal en recibida en banda base

En primer lugar, se transmite sólo una trama de datos a diferencia del resto de pruebas en las que el *Transmisor* generaba tramas de manera continua.

Por otra parte, hay que suponer que el *Receptor* es capaz de sincronizar la trama a la primera, es decir, que los datos llegarán sin errores y el bloque de sincronismo es capaz de conseguir el inicio de la trama sin necesidad de desechar ninguna.

Una vez sincronizada, se encenderán dos LEDs de la placa y se comenzará con la transmisión de los datos recibidos. Estos datos, formados por la información recibida exceptuando los STS, serán transmitidos de nuevo de forma continua con el fin de poder visualizarlos mediante el oscilospio.

En la imagen de la figura 6.8, se puede ver la repetición de la trama de forma continua, comprobando además que la duración de la misma es prácticamente la misma a la vista en las anteriores secciones de este capítulo.

6.5. Ocupación de recursos

Finalmente, en la figuras 6.9 y 6.10 se muestran los recursos ocupados del bloque *Transmisor* y *Receptor* respectivamente.

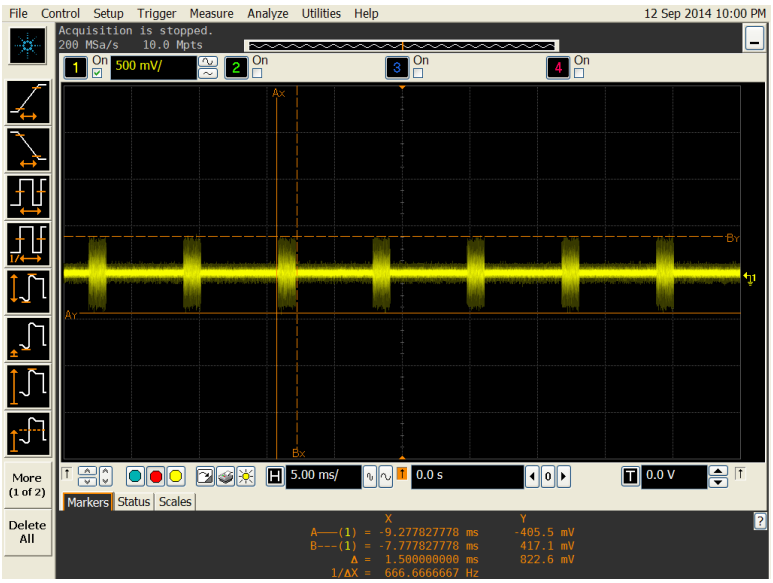


Figura 6.8: Pruebas de sincronismo

Como se puede comprobar, los recursos del transmisor se han conseguido reducir con respecto a las anteriores fases del proyecto, consiguiendo disminuir el número de estructuras de datos (LUTs), de Flip Flops y de registros (Slices), además de mantener el comportamiento lógico de los últimos.

En cuanto al receptor, la ocupación es algo mayor pero se sigue manteniendo un número de recursos utilizados bajo.

Resumiendo, en ambos casos la ocupación es baja por lo que la utilización de los recursos disponibles se ha hecho de manera eficiente.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	4,185	30,720	13%	
Number of 4 input LUTs	5,238	30,720	17%	
Number of occupied Slices	3,795	15,360	24%	
Number of Slices containing only related logic	3,795	3,795	100%	
Number of Slices containing unrelated logic	0	3,795	0%	
Total Number of 4 input LUTs	5,537	30,720	18%	
Number used as logic	4,923			
Number used as a route-thru	299			
Number used as 16x1 RAMs	16			
Number used as Shift registers	299			
Number of bonded IOBs	211	448	47%	
IOB Dual-Data Rate Flops	4			
IOB Master Pads	12			
IOB Slave Pads	12			
Number of BUFG/BUFGCTRLs	9	32	28%	
Number used as BUFGs	8			
Number used as BUFGCTRLs	1			
Number of FIFO16/RAMB16s	27	192	14%	
Number used as RAMB16s	27			
Number of DSP48s	30	192	15%	
Number of DCM_ADVs	2	8	25%	
Number of RPM macros	3			
Average Fanout of Non-Clock Nets	2.99			

Figura 6.9: Recursos ocupados por el bloque *Transmisor*

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	5,871	30,720	19%	
Number of 4 input LUTs	7,997	30,720	26%	
Number of occupied Slices	5,329	15,360	34%	
Number of Slices containing only related logic	5,329	5,329	100%	
Number of Slices containing unrelated logic	0	5,329	0%	
Total Number of 4 input LUTs	8,360	30,720	27%	
Number used as logic	7,661			
Number used as a route-thru	363			
Number used as 16x1 RAMs	32			
Number used as Shift registers	304			
Number of bonded IOBs	242	448	54%	
IOB Dual-Data Rate Flops	4			
IOB Master Pads	12			
IOB Slave Pads	12			
Number of BUFG/BUFGCTRLs	11	32	34%	
Number used as BUFGs	9			
Number used as BUFGCTRLs	2			
Number of FIFO16/RAMB16s	57	192	29%	
Number used as RAMB16s	57			
Number of DSP48s	159	192	82%	
Number of DCM_ADVs	3	8	37%	
Number of RPM macros	6			
Average Fanout of Non-Clock Nets	2.66			

Figura 6.10: Recursos ocupados por el bloque *Receptor*

Capítulo 7

Conclusiones

A lo largo de este capítulo se irá valorando el trabajo realizado en este Proyecto Fin de Carrera desde los resultados obtenidos, verificando si se han cumplido los objetivos marcados, hasta las posibles líneas futuras con el fin de dar continuidad al trabajo realizado, pasando además por las dificultades encontradas.

En este trabajo se ha desarrollado un sistema de comunicaciones OFDM, desde su diseño hasta su implementación en un dispositivo SFF SDR. Como se ha ido viendo, se empezó con un diseño teórico que se fue modificando con el fin de adaptarlo al hardware haciendo uso del lenguaje VHDL.

En primer lugar, se ha desarrollado un transmisor siguiendo con la línea marcada en las fases anteriores del proyecto, mejorando su diseño incluyendo nuevas funcionalidades. Uno de los principales cambios ha sido la FFT. Como se verá a continuación en la sección 7.2, fue este *IP Core* el que ha dado más problemas en esta parte del proyecto debido a las distintas versiones de *Xilinx*.

Sin embargo, a pesar de este inconveniente, el transmisor implementado ha dado resultados satisfactorios en cuanto a recursos ocupados. Además como se vio en el capítulo 6, se ha conseguido la generación de tramas OFDM con unos tiempos de transmisión y de procesamiento esperados.

Por otra parte, también se han conseguido los objetivos marcados para el receptor. No sólo se ha podido comprobar mediante simulación que se han realizado las operaciones inversas realizadas por el transmisor, sino que también se ha conseguido una buena sincronización y una ecualización, gracias a la cual se han corregido posibles variaciones de amplitud y fase de nuestros símbolos transmitidos.

A diferencia del transmisor, comprobar el funcionamiento del receptor en el dispositivo es más complicado ya que no se disponía de ningún puerto de salida en el que tomar la información demodulada. Lo único que se ha podido comprobar ha sido el sincronismo mediante un algoritmo que mostraba que la sincronización funcionaba.

Aunque en la memoria de este proyecto no se ha centrado en los dispositivos empleados, se ha conseguido obtener una familiarización entre todos los módulos del dispositivo. Esta coordinación entre los módulos junto con el estudio también de todas las librerías de VHDL ha sido llevada a cabo y es importante mencionarla como uno de los objetivos alcanzados.

Finalmente, se puede decir que se han logrado los objetivos marcados a pesar de todas las dificultades encontradas en el proceso que serán explicadas más adelante.

7.1. Líneas futuras

En esta sección se enumerarán y describirán las posibles mejoras y ampliaciones que pueden ser llevadas a cabo:

- Diseñar un transmisor y receptor capaz de soportar distintas constelaciones QAM.
- Desarrollar una estimación de canal utilizando esta vez un criterio MMSE con el fin de conseguir mejores resultados.
- Utilizar los conceptos del sistema OFDM para desarrollar sistemas de comunicaciones WiMAX, UMTS, LTE, etc
- Estudio de la BER y la SNR en función de distintos parámetros en el caso de poder tener acceso a la información demodulada.
- Realizar el cálculo del FFT en paralelo, es decir, calcular varias FFT al mismo tiempo con el fin de reducir la latencia en la generación de la trama OFDM.

7.2. Dificultades

A continuación se mostrarán las principales dificultades encontradas en el desarrollo de este Proyecto Fin de Carrera.

En primer lugar, la mayor dificultad encontrada es la falta de información sobre el dispositivo SFF SDR. A pesar de tener un manual de usuario, este resulta en numerosas ocasiones insuficiente, teniendo que recurrir a otras fuentes para intentar hallar la respuesta.

Este proyecto daba continuidad a los trabajos realizados en [3], [4], teniendo que unir ambos trabajos. A pesar de que todo se encontraba bien explicado en sus respectivas memorias, hay detalles que se pueden escapar.

Por otra parte, la familiarización con los *IP Cores* de *Xilinx* es bastante costosa. La información disponible es en ocasiones algo complicada de entender, de ahí que para cada *IP Core* utilizada se generase un código auxiliar con el fin de entender el funcionamiento de todas sus interfaces de entrada y de sus distintas propiedades a la hora de crearlos.

Fue precisamente el *IP Core* FFT el que más problemas dio debido a las distintas versiones de *Xilinx* en las que se había realizado este proyecto. Debido a este desajuste entre las versiones, el código realizado en las fases anteriores no sintetizaba impidiendo así poder realizar pruebas.

Aunque para el lector, estas dificultades puedan parecer pequeñas y normales como en cualquier otro trabajo, lleva mucho tiempo descubrirlas, retardando así el desarrollo del mismo.

Glosario

A

ADC *Analog-to-Digital Converter*, p. 9.

ADSL *Asymmetric Digital Subscriber Line*, p. 1.

AGC *Automatic Gain Control*, p. 2.

B

BER *Bit Error Ratio*, p. 5.

D

DAC *Digital-to-Analog Converter*, p. 9.

DFT *Discrete Fourier Transform*, p. 10.

DSP *Digital Signal Processing*, p. 1.

DVB-T *Digital Video Broadcasting-Terrestrial*, p. 1.

F

FDM *Frequency Division Multiplexing*, p. 6.

FFT *Fast Fourier Transform*, p. xi.

FIR *Finite Impulse Response*, p. 15.

FPGA *Field Programmable Gate Array*, p. 1.

I

ICI *InterCarrier Interference*, p. 8.

IDFT *Inverse Discrete Fourier Transform*, p. 10.

IF *Intermediate Frequency*, p. 85.

IFFT *Inverse Fast Fourier Transform*, p. 7.

IQ *Inphase-Quadrature*, p. 9.

ISI *InterSymbol Interference*, p. 8.

L

LS *Least Squares*, p. 14.

LTE *Long Term Evolution*, p. 96.

LTS *Long Training Sequence*, p. 12.

M

MMSE *Minimum Mean Square Error*, p. 14.

O

OFDM *Orthogonal Frequency Division Multiplexing*, p. xi.

P

PC *Prefijo Cíclico*, p. xi.

PFC *Proyecto Final de Carrera*, p. 5.

PLCP *Physical Layer Convergence Protocol*, p. 12.

Q

QAM *Quadrature Amplitude Modulation*, p. xi.

R

RAM *Random Access Memory*, p. xii.

S

SFF SDR *Small Form Factor Software Defined Radio*, p. 1.

SISO *Single Input Single Output*, p. 83.

SNR *Signal to Noise Ratio*, p. 74.

STS *Short Training Sequence*, p. xii.

T

TFG *Trabajo Fin de Grado*, p. 73.

U

UMTS *Universal Mobile Telecommunicatins System*, p. 96.

V

VHDL *VHSIC Hardware Description Language*, p. 2.

W

WiFi *Wireless Fidelity*, p. 1.

WiMAX *Worldwide interoperability Microwave Access*, p. 1.

WLAN *Wireless Local Area Network.*, p. xi.

Anexos

Anexo A

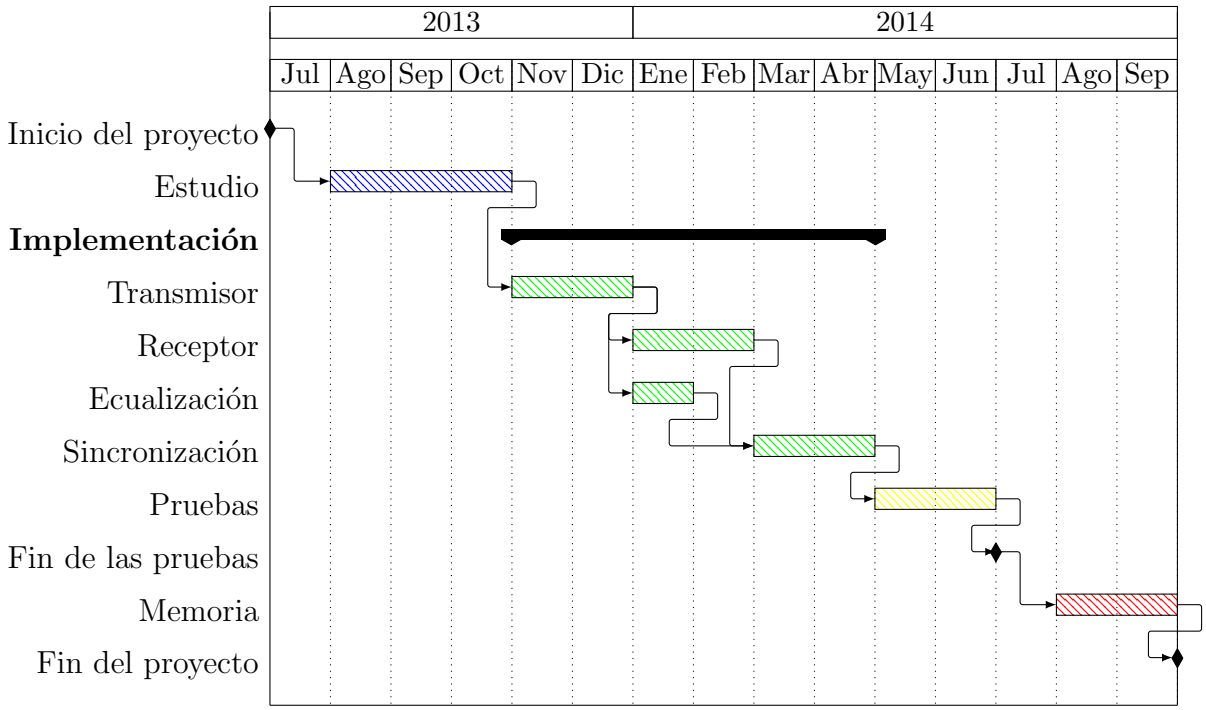
Presupuesto

Tal y como se ha visto a lo largo de esta memoria, este proyecto implementa un sistema de comunicaciones OFDM en un dispositivo SFF SDR, mostrando tanto su diseño, simulación e implementación en el dispositivo, intentando hacerlo de una forma sencilla.

Asimismo, se ha dividido el desarrollo del proyecto en las siguientes fases:

- Familiarización con el dispositivo. Estudio de la OFDM, de anteriores fases del proyecto, manuales de usuario de la placa y de las herramientas software empleadas.
- Desarrollo y simulación del sistema de comunicaciones. Esta fase se puede dividir, a su vez, en otras cuatro:
  - Transmisor (Capítulo 2)
  - Receptor (Capítulo 3)
  - Sincronismo (Capítulo 4)
  - Ecualización (Capítulo 5)
- Realización de pruebas sobre la placa.
- Elaboración de la memoria del presente proyecto.

De forma que se crea el siguiente diagrama de Gantt detallando el desarrollo cronológico del proyecto:



Y finalmente, el presupuesto considerando todos los costes posibles: personas involucradas en el desarrollo, luz eléctrica, equipos empleados, documentación, etc.

En la tabla A.1 se muestran todos los gastos relativos al personal. En total, la duración del proyecto ha sido de 1.529 horas, de las cuales un 10 % han sido compartidas con el tutor del mismo. Suponiendo que el coste de un ingeniero junior se sitúa en torno a los 15 €/h y el de un ingeniero senior en los 40 €/h, el gasto de personal asciende a un total de 28.495€.

Personal	Horas empleadas	€/hora	Importe(€)
Ingeniero Junior	1.529	15	22.935
Ingeniero Senior	139	40	5.560
		Total	28.495

Cuadro A.1: Costes de personal

Por otra parte tenemos los costes de material que se resumen en la tabla A.2. En este cálculo se ha tenido en cuenta una vida útil de todos los materiales

involucrados en el proyecto de 5 años. Para el cálculo de la amortización de los mismos se ha seguido la siguiente fórmula:

$$\frac{A}{B} \times C \tag{A.1}$$

donde A es el número de meses desde la fecha de facturación en que el equipo ha sido utilizado, B es el periodo de depreciación y C el coste del equipo sin IVA.

Descripción	Coste(€)	Dedicación (meses)	Importe (€)
Ordenador gama media	1.000	14	233,33
SFF SDR y licencias	8.372,20	14	1.953,51
Osciloscopio Infiniium DSO90604A	41.967,45	14	9.792,41
Otros gastos	6.000	14	6.000
		Total	17.979,25

Cuadro A.2: Costes materiales

En la tabla A.2 se ha incluido también otros costes directos del proyecto como son el local, luz eléctrica, viajes, etc.

Por último, sumando los costes de personal, los costes de material y los costes indirectos (20 % de la suma de los costes de personal y material) obtendremos el presupuesto mostrado en la tabla A.3.

Concepto	Presupuesto(€)
Costes de personal	28.495
Costes de material	17.979,25
Costes indirectos	9.294,85
Total	55.769,1

Cuadro A.3: Coste total

Así, el presupuesto total de este proyecto asciende a la cantidad de 55.769,1 euros.

Leganés, a 15 de Septiembre de 2014.

El ingeniero proyectista.

Bibliografía

[1] IEEE Standars Association. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. March 2012.

[2] Xilinx. *Fast Fourier Transform v7.1 Product Specification*, March 2011.

[3] Rony Fonseca Arboleda. Implementación de un sistema de comunicación en un dispositivo radio. Master's thesis, Universidad Carlos III de Madrid, 2012.

[4] Carlos Valverde Muñoz. Implementación de un sistema de OFDM en un dispositivo SFF SDR. Master's thesis, Universidad Carlos III de Madrid, 2010.

[5] V. P. Gil Jiménez, J. Fernández-Getino García, F. J. González Serrano, and Ana García Armada. Design and Implementation of Synchronization and AGC for OFDM-based WLAN Receivers. *IEEE Transactions on Consumer Electronics*, 50(4):1016–025, Noviembre 2004.

[6] Xilinx. *Block Memory Generator v7.3 Product Specification*, December 2012.

[7] Xilinx. *Divider Generator v3.0 Product Specification*, March 2011.

[8] Emilio López Álvaro. Simulación de MIMO OFDM en el downlink de LTE. Master's thesis, Universidad Carlos III de Madrid, 2013.

[9] María Martín Fernández. Pruebas de transmisión con equipos software radio. Master's thesis, Universidad Carlos III de Madrid, 2012.

[10] Lyrtech. *Small Form Factor SDR Evaluation Mode/ Development Platform. User's guide*, October 2007.

[11] Lyrtech. *Small Form Factor SDR Evaluation Mode/ Development Platform. DSP API*, January 2007.

[12] Lyrtech. *SFF SDR DP. Quick Start Quide*, October 2007.

[13] Lyrtech Development Platform Command Shell User's Guide v1.6. *Lyrtech*, October 2007.

[14] Virtex-4 Libraries Guide for HDL Designs. *Xilinx*, 2007.

[15] Code Composer Studio Development Tools v3.3. Getting Started Guide. *Texas Instrument*, October 2007.

[16] ADACMaster III. User's guide. *Lyrtech*, November 2010.

[17] Amontec. *Using Xilinx IP Cores*, June 2001.

[18] A. Artés Rodriguez, F. Pérez González, J. Cid Sueiro, R. López Valcarce, C. Mosquera Nartallo, and F. Pérez Cruz. *Comunicaciones Digitales*. Pearson College, 2007.

[19] Miguel Cordero Limón. Técnicas de estimación de canal en la capa física Wireless MAN-OFDM de la norma IEEE 802.16e. Master's thesis, Universidad de Sevilla, 2009.

[20] Víctor P. Gil Jiménez. *Algoritmos en Transmisión y Recepción para OFDM en entornos Multi-usuario*. PhD thesis, Universidad Carlos III de Madrid, 2005.